Тр 40н.

Д. ФЕРРАРИ

ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Д. ФЕРРАРИ

ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

COMPUTER SYSTEMS PERFORMANCE EVALUATION

Domenico Ferrari

Computer Science Division
Department of Electrical Engineering
and Computer Sciences
University of California, Berkeley

Prentice-Hall, Inc , Englewood Cliffs,
New Jersey 07632
1978

Д. ФЕРРАРИ

ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Перевод с английского А. И. Горлина,
Ю. Б. Котова
и Л. В. Ухова

под редакцией В. В. Мартынюка

ИЗДАТЕЛЬСТВО «МИР» МОСКВА 1981

681.142.2
Ф43

Феррари Д.

Ф43 Оценка производительности вычислительных систем: Пер. с англ. А. И. Горлина, Ю. Б. Котова и Л. В. Ухова / Под ред. В. В. Мартынюка.— М.: Мир, 1981.
576 с.

В книге известного американского специалиста Д. Феррари изложен единый подход к оценке эффективности и производительности оборудования и программного обеспечения ЭВМ. Подробно разобраны более 70 примеров оценки производительности систем, даны практические рекомендации по методике измерений и моделирования систем. Каждая глава сопровождается упражнениями для самостоятельной работы.
Для системных и прикладных программистов, аспирантов и студентов университетов.

Ф $\frac{20204\text{-}031}{041(01)\text{-}81}$ 31-81, ч. 1 2405000000 681.142.2

Редакция литературы по математическим наукам

© 1978 by Prentice-Hall, Inc. Englewood Cliffs, New Jersey 07632

© Перевод на русский язык, «Мир», 1981

Д. Феррари
ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Научный редактор Л. Н. Бабынина Мл. редактор Р. И. Пяткина Художник С. А. Бычков Художественный редактор В. И. Шаповалов Технический редактор З. И. Резник Корректор Н. А. Гиря

ИБ № 2301

Сдано в набор 26-01.81. Подписано к печати 11.06.81. Формат 60×90¹/₁₆. Бумага типографская № 1. Гарнитура латинская. Печать высокая. Объем 18 бум. л. Усл. печ. л. 36. Усл. кр. отт. 36. Уч.-изд. л. 43,76. Изд. № 1/0859. Тираж 15 000 экз. Заказ № 2476. Цена 3 р. 40 к.

ИЗДАТЕЛЬСТВО «МИР» Москва, 1-й Рижский пер., 2

Ордена Октябрьской Революции и ордена Трудового Красного Знамени Первая Образцовая типография имени А. А. Жданова Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. Москва, М-54, Валовая, 28

От редактора и переводчиков

Интерес к эффективности использования вычислительных систем закономерно возник в последнее десятилетие в связи со значительным увеличением разнообразия пользователей. Отсутствие в отечественной литературе специального курса оценки производительности таких систем стало восприниматься как пробел в подготовке специалистов по машинной математике. Восполнить этот пробел поможет книга известного специалиста в данной области Д. Феррари, созданная им на основе курса лекций для студентов университета в Беркли. Она является введением в круг идей и основных методических приемов такой оценки.

Проблема оценки производительности интересует практически всех специалистов, имеющих дело с современной вычислительной техникой. Для потребителей вычислительных комплексов она приобретает форму выбора варианта с наилучшей производительностью из множества предлагаемых изготовителями. При этом критерии выбора системы для научно-исследовательских работ не совпадают с критериями выбора в управленческих и коммерческих целях. Для групп, занятых эксплуатацией и развитием систем, возникает задача планирования усовершенствований с целью достичь максимального роста производительности при минимальном объеме изменений. Разработчики встречаются с задачей прогноза круга работ, предстоящих создаваемой ими системе, и проектирования варианта, наилучшим образом обеспечивающего этот круг работ. Таким образом, цель исследования в значительной степени определяет смысл и специфику оценки производительности.

Самый очевидный путь оценки производительности — измерение характеристик реально существующей системы на реально решаемых задачах — является и самым трудоемким. Кроме того, он не дает ответа на вопрос об изменении оценочных характеристик при небольших структурных и организационных изменениях или при изменении характеристик входного потока. Более плодотворным оказывается метод моделирования. Он позволяет найти достаточное количество динамических характеристик системы, описанной статически с необходимой степенью подробности. В ряде случаев для получения ориентировочных результатов можно воспользоваться аналитическим методом исследования. Однако, как правило, реальные системы из-за их сложности не поддаются аналитической оценке. В книге подробно обсуждены

возможности применения всех этих методов (измерение, моделирование, аналитическое исследование) к трем практическим проблемам (выбор, усовершенствование и разработка).

В целом книга Д. Феррари представляет собой удачный курс методов оценки производительности вычислительных систем. Пониманию излагаемых методов помогают многочисленные подробно разобранные примеры оценки. Несомненную практическую ценность представляют полезные рекомендации, которые сопровождают изложение материала на протяжении всей книги.

Специалистам по вычислительной технике и математическому обеспечению ЭВМ книга может помочь выявить резервы повышения производительности, а многочисленным пользователям ЭВМ она позволит более отчетливо понять, какой ценой достигаются те или иные возможности, которые предоставляются или могли бы быть предоставлены доступными им вычислительными системами.

Главы 1—3, 6—8 и предисловие перевел Ю. Б. Котов, главы 4 и 5 — А. И. Горлин, главу 9 — Л. В. Ухов.

Алессандре и Джулиаракеле
за их любовь, терпение и поддержку

О любом деле можно сказать, что само по себе оно не бывает ни прекрасным, ни безобразным. Например, все, что мы делаем сейчас, пьем ли, поем ли или беседуем, прекрасно не само по себе, а смотря по тому, как это делается, как происходит: если дело делается прекрасно и правильно, оно становится прекрасным, а если неправильно, наоборот, безобразным.

Платон «Пир» (перев. С. Апта)

Говорят, что все влюбленные клянутся совершить больше, чем они способны, что они хвастают за десятерых, а не делают и десятой доли того, что может сделать один; говорят голосом льва, а поступают как зайцы. Разве они не чудовища?

Шекспир «Троил и Крессида»
(перев. Т. Гнедич)

И школа светская, хороший тон внушая,
Не меньше учит нас, чем книга пребольшая.

Мольер «Школа мужей»
(перев. В. Гиппиуса)

Предисловие

Компьютеры — это машины, изготовленные людьми, и они относятся поэтому к области техники. Они предназначены для удовлетворения потребностей отдельных пользователей и организаций в переработке информации. При создании компьютеров инженеры стремятся к заданной производительности при определенных ограничениях на стоимость. Люди, разрабатывающие программы для компьютеров, выполняют аналогичную инженерную работу.

Существенным аспектом любой инженерной деятельности является оценка объекта разработки. Техническую систему оценивают проектировщики, изготовители, покупатели, администраторы и пользователи. Одним из критериев оценки системы (причем очень важным критерием) является производительность. Эта книга посвящена оценке производительности вычислительных систем. Тема книги заинтересует (или должна заинтересовать) многих специалистов, работающих с вычислительными машинами. Разработчики систем, обслуживающий персонал и руководители подразделений всех уровней, системные аналитики, разработчики программ и пользователи вычислительных машин ежедневно сталкиваются с проблемами, решение которых можно существенно облегчить и улучшить, если использовать технические приемы и средства оценки производительности.

Для всех вычислительных установок приходится решать проблемы, тесно связанные с оценкой производительности, например проблемы выбора системы, проектирования ее конфигурации, настройки, развития систем, учета и оценки стоимости, управления работой системы, краткосрочного и долгосрочного планирования. Практически все организации, пользующиеся вычислительными системами, должны, принимая важные решения, учитывать требования производительности. Если система не обеспечивает ожидаемой производительности или не обеспечивает минимального отношения «стоимость/производительность», то это означает растрату ресурсов впустую, что нежелательно.

Оценка производительности заинтересовывает все большее число исследователей и преподавателей по мере признания ее практической важности. Такая оценка может рассматриваться как применение научного подхода к изучению вычислительных систем. Однако курс оценки производительности изучается отдельно от курса проектирования систем, что является признаком неудовлетворительной ситуации. В других, более традиционных, отраслях техники редко удается увеличить производительность спроектированной и изготовленной системы примитивными методами в заметное число раз. С вычислительными системами это случается довольно часто. Поскольку оценка производительности является составной частью проектирования любого вида техники, то крайне мало инженеров специализируется на оценке производительности автомобилей, самолетов, мостов, промышленных процессов, электротехнических устройств.

Теория вычислительных систем еще недостаточно разработана. Она пока еще не может основываться на количественных закономерностях, подобных законам, устанавливающим научные основы других видов техники: законам механики, электромагнетизма, термодинамики и т. д. Поэтому конечной целью изучения проблем оценки производительности должно быть внедрение результатов в методы проектирования, изготовления и использования. Иначе говоря, конечная цель состоит в том, чтобы сделать ненужным специальное исследование производительности самой по себе.

Можно ожидать, что в отдаленном будущем количественный подход войдет в основу курсов машинной математики и в учебные программы институтов, сделав тем самым необязательными специальные курсы оценки производительности. Однако, прежде чем это будет достигнуто, потребуются обширные и успешные исследования и даже радикальное изменение мышления, что гораздо важнее, но и гораздо труднее. Различие между существующим положением дел и упомянутой выше целью лучше всего, вероятно, иллюстрирует почти исключительно качественный и описательный подход, принятый ныне в преподавании курса организации вычислительных систем. В такой ситуации необходимы специальные курсы, посвященные оценке производительности. Они должны познакомить студентов с очень важной точкой зрения на вычислительные системы, дать общую картину этой быстро растущей области, ее основных тем, проблем, известных подходов к этим проблемам и их решений.

Данная книга написана на основе одного из таких курсов, несколько раз прочитанного студентам университета в Беркли. Уровень курса (а следовательно, и книги) выбран таким, чтобы он был доступен как студентам старших курсов, специализирующимся в области машинной математики, так и аспирантам первого года обучения. Необходимыми условиями для понимания этой книги является некоторое знакомство с материалом обычного двухсеместрового курса организации вычислительных или операционных систем, а также с принципами и основными методами программирования и основами дискретной математики, статистики и теории вероятностей. У специалистов в области машинной математики не возникнет проблем с первыми двумя необходимыми условиями (вычислительные системы и программирование), а ссылки на учебники, излагающие математический и статистический материал, необходимый для работы с этой книгой, они найдут в библиографии.

В книге основное внимание уделено принципиальным сторонам

методов оценки производительности и проблемам, которые, как мне кажется, должны рассматриваться в первую очередь. Однако, я не пренебрегал и теми информативными аспектами, которые могут помочь читателю получить исчерпывающее представление о состоянии дел в исследованиях и в коммерческом мире. Итоговая картина является, разумеется, глубоко субъективной. Я выделял те методы, точки зрения и подходы, которые, по-моему, будут становиться все важнее в ближайшем будущем, а не стремился дать точный моментальный снимок существующей ситуации. Методы и средства, заимствованные из других областей, представлены в книге так, как будто они были изобретены для оценки вычислительных систем, и их характеристики обсуждаются только с этой точки зрения. Я совершил также ряд других больших и малых грехов, которые читатель (надеюсь, снисходительный) скоро обнаружит.

Книгу объединяет очень прагматическая тема: *исследование по оценке* рассматривается как набор процедур, цель которых — накопить информацию об оцениваемой системе, позволяющую ответить на определенные вопросы о производительности. Таким образом, эта книга описывает, во-первых, исследования по оценке, необходимые для решения основных проблем технологии вычислительных систем, и, во-вторых, методы, которые должны использоваться в этих исследованиях.

Во вводной главе 1 характеризуется подход к рассмотрению вычислительных систем. В этой главе рассматриваются понятие производительности, наиболее важные индексы производительности, широко распространенные способы классификации проблем и методов оценки.

Последующие три главы посвящены основным методам и средствам оценки, большинство которых широко применяется в других областях науки и техники. Здесь содержится некоторое минимальное количество предварительной информации, а в основном рассматривается применение этих методов к вычислительным системам. Глава 2 описывает измерительные исследования, измерители, планирование экспериментов и применение методов эмпирического моделирования к проблеме интерпретации результатов эксперимента. В главе 3 описано моделирование вычислительных систем. Рассматриваются формулировка, построение модели, ее калибровка и контроль, а также планирование экспериментов над имитационной моделью и интерпретация результатов моделирования. Глава 4 посвящена аналитическому моделированию. Изучены как детерминированная, так и вероятностная модели систем, причем особый акцент делается на применении методов моделирования сетей с очередями для анализа производительности. В главе 5 рассматривается описание рабочей нагрузки — фундаментальная проблема для любой оценки. Недостаточность наших знаний в этой области является, вероятно, самым серьезным препятствием для прогресса в оценке производительности. Этой теме посвящена отдельная глава, поскольку основные проблемы описания рабочей нагрузки в принципе одинаковы для всех методов. В данной главе

эти проблемы исследуются вместе с известными подходами к моделированию рабочей нагрузки.

Ознакомившись с методами, составляющими существенную часть любой работы по оценке, читатель должен быть готов применить их к различным типам задач. Три обширных класса проблем исследованы в последующих трех главах. В главе 6 описаны технические аспекты выбора компьютера и оценивается пригодность известных методов и измерителей применительно к этой задаче. Глава 7 посвящена повышению производительности, области, в которой были достигнуты наиболее поразительные успехи. Мы надеемся, что важность этой области будет убывать по мере появления лучших методов проектирования систем и установок. В главе 8 обсуждается роль оценки производительности в проектировании систем. Методы, приведенные в начале этой книги, применяются здесь к различным проблемам проектирования.

Наконец, в главе 9 рассмотрена оценка производительности программ. Индексы производительности, методы оценки и проблемы, относящиеся к производительности программ, даны в соответствии с последовательностью изложения.

Эту книгу можно правильно охарактеризовать как общее введение в оценку производительности. Каждую главу можно было бы рассматривать как введение или обширное резюме к отдельной книге, которую можно было бы написать по данному предмету. Книгу можно использовать как учебник по курсам оценки производительности вычислительных машин, проектирования вычислительных и операционных систем. Данная книга (возможно, дополненная периодикой и другими книгами) может также служить руководством для специальных курсов по измерениям, моделированию вычислительных систем, методам выбора компьютера, методам настройки и оценки программного обеспечения.

Много специалистов и организаций оказали мне существенную помощь в создании и подготовке этой книги. Необходимые основы были заложены теми, кого я могу назвать моими учителями. Прежде всего я должен назвать своих родителей, прививших мне свой взгляд на жизнь, и моих студентов в университете в Беркли, интерес которых к предмету был для меня постоянным источником вдохновения. Я также благодарен ряду моих коллег по отделу электротехники и машинной математики за их моральную и научную поддержку и за создание творческой обстановки. Некоторые специалисты стимулировали мою работу своими исследованиями и своим интересом к этой книге. Я в долгу перед всеми авторами, перечисленными в библиографии, и перед многими другими специалистами в области оценки производительности. Моя особая благодарность обращена ко всем, давшим мне ценные советы, особенно рецензентам Гарольду Хиту, Дэниелу Сивиореку, Стиву Кимблтону, Фрэнку Палермо и Феликсу Лэму. Крайне полезными на протяжении всей моей работы были воодушевление и издательская поддержка Карла Карлстрома. Перепечатку рукописи проделали с большим вниманием, редким терпением и превосходными результатами

Рут Сузуки, Эдит Персер и Мэри Энн Рэч. Наконец, я хотел бы поблагодарить птиц, белок, деревья и цветы в моем саду в Беркли, в обществе которых была написана бóльшая часть этой книги.

Правление Калифорнийского университета содействовало подготовке книги, предоставив мне оплаченный творческий отпуск в течение семестра, а также финансовую поддержку. Университетам Пизы и Генуи и Миланскому политехническому институту за их поддержку также следует воздать должное. Некоторые работы, приведенные в главе 2, финансировались Управлением перспективного планирования научно-исследовательских работ по контракту DAHC15-70-C-0274. Отдельные материалы, приведенные в главе 9, получены отделением вычислительных исследований Национального научного фонда по контракту DCR74-18375 и исследовательским отделением фирмы IBM.

За все ошибки и пробелы я никому, кроме себя, не обязан.

Доменико Феррари

1 Подход к оценке производительности

1.1. ВВЕДЕНИЕ

Оценка производительности играет существенную роль во всех отраслях техники. Каждая создаваемая система (автомобиль, здание, станок, промышленное предприятие) должна удовлетворять заранее заданным *спецификациям производительности*. Для получения систем, удовлетворяющих этим спецификациям, конструкторы используют специальные методики проектирования и процедуры оценки. Некоторые приемы и методы оценки могут также использовать потенциальные и реальные пользователи системы, когда им необходимо принять некоторое решение о системе (например, покупка, аренда, продажа, модификация).

Природа спецификаций производительности обычно меняется в зависимости от типа системы и от точки зрения оценщика. Каждая система должна выполнять некоторую функцию или набор функций. Таким образом, наиболее важное условие, которому она должна удовлетворять, состоит в правильном выполнении ею своей функции (или функций). Правильность есть обязательная предпосылка для всех остальных требований. Ее подтверждение, вообще говоря, не зависит от субъективного взгляда, хотя для большинства систем относительно просто достичь согласия по вопросу, работают ли они. Например, транспортное средство должно двигаться, мост не должен разрушаться, а усилитель должен усиливать электрический сигнал.

Спецификации производительности, которым должна отвечать система, чтобы быть приемлемой для пользователей, характеризуют, *насколько хорошо* система выполняет свои функции. Выбор требований производительности обычно более субъективен, чем выбор спецификаций правильности. Однако первые часто выражаются в количественных терминах, так что понятийно их проверка не сложнее, чем последних. Например, максимальная скорость автомобиля с грузом 1000 фунтов должна быть более 100 миль в час; к.п.д электрического трансформатора должен быть более 90% в заданном диапазоне нагрузок. Уж что действительно субъективно и очень сильно зависит от точки зрения оценщика, то это относительное значение, придаваемое различным аспектам производительности; например, скорость автомобиля часто имеет второстепенное значение по сравнению с его надежностью, долговечностью, расходом топлива, комфортом и удобством управления.

Различие между аспектами правильности и производительности часто оказывается не слишком резким. В самом деле, правильность можно было бы рассматривать как один из аспектов (действительно, важный аспект) производительности. Тем не менее в этой книге мы будем использовать термин *производительность* именно для того, чтобы показать, *насколько хорошо* работает система, подразумевая, что она работает правильно. Читатель, конечно, понимает, что правильности очень редко бывает достаточно, чтобы сделать систему приемлемой. Иными словами, выполнение требований производительности, как и спецификаций правильности, очень часто одинаково важно для определения пригодности системы для некоторого применения. Например, часы, стрелки которых движутся, но точность которых слишком низка для их предполагаемого использования, вообще бесполезны.

В соответствии с предложенным выше определением, производительность можно интерпретировать как технический эквивалент экономического понятия *ценности* (потребительской стоимости). Иначе говоря, производительность есть то, что делает систему ценной для пользователя. Подобно ценности, производительность является только одной из двух сторон реальности в экономическом мире, другой же стороной являются затраты пользователя (цена). Решения относительно систем, будь то решение о разработке, изготовлении, приобретении или модификации, почти всегда диктуются обоими соображениями: как производительностью, так и затратами.

Концепция производительности субъективна. Однако, как будет показано в разд. 1.4, часто удается перевести субъективные определения производительности в чисто технические термины, которым иногда может быть придана количественная форма и которые, следовательно, можно оценить объективно. Таким образом, работа по *оценке производительности* может (а в этой книге будет) рассматриваться как техническая деятельность, имеющая целью измерение производительности. Мы не будем непосредственно касаться аспектов затрат на систему, производительность которой предстоит оценивать. Тем не менее читателю следует постоянно помнить, что затраты и производительность можно разделять только в педагогических целях, реальная же оценка производительности должна всегда сопровождаться некоторой формой определения затрат.

В этой главе мы введем общие понятия и определения, которые будут использованы в последующих главах этой книги, описывающих в деталях различные стороны деятельности по оценке производительности в области вычислительных систем. Эта деятельность в целом называется *исследование оценки производительности*. Типы систем, оценкой производительности которых мы будем интересоваться, обсуждаются в разд. 1.2. Этапы, из которых должно состоять исследование, обсуждены в разд. 1.3. В разд. 1.4 мы будем оперировать критериями и индексами производительности и введем существенно более строгие определения производительности, чем обсужденное выше.

В этой книге мы в основном будем иметь дело с оценкой производительности в соответствии с определением, данным в разд. 1.4.

В разделе 1.5 будет введена классификация, которую мы примем для задач оценки; этот раздел можно рассматривать как введение к гл. 6, 7 и 8. Наконец, методы и приемы оценки, которые будут описаны в гл. 2, 3, 4 и 5 впервые вводятся в разд. 1.6.

1.2. ОБЪЕКТЫ РАБОТ ПО ОЦЕНКЕ

В вычислительной технике оценка производительности менее развита, но не менее важна, чем в других, более старых отраслях техники. Все изделия вычислительной техники предназначаются для выполнения определенных функций, относящихся к обработке информации. С технической, экономической и общественной точек зрения очень важно, насколько хорошо эти системы выполняют свои задачи. Их оценка, как будет показано в этой книге, является привлекательным, но ни в коей мере не легким делом, по крайней мере при современном состоянии наших знаний.

В вычислительной технике существует много типов систем, выполняющих различные функции и имеющих разную природу. Например, есть вычислительные установки, вычислительные системы, вычислительные сети, компоненты систем, операционные системы, программы, языки программирования, трансляторы. Производительность этих систем должна оцениваться их разработчиками, изготовителями, администрацией, обслуживающим персоналом и пользователями. Эта книга посвящена оценке производительности вычислительных систем и (в гл. 9) программ. Оценка других типов систем рассматривается только в тех аспектах, которые непосредственно относятся к оценке вычислительных систем и программ. Например, методы оценки для центральных процессоров (ЦП) не выделяются как отдельный предмет, а используются только тогда, там и в такой мере, насколько это необходимо для постановки и решения задач оценки производительности вычислительных систем или программ.

Удобное для наших целей определение *программы* будет дано в разд. 9.1. А теперь, в оставшейся части данного раздела, мы введем предварительное определение вычислительных систем и обсудим их черты с точки зрения оценки производительности.

Вычислительная система (которая для краткости будет часто называться просто *системой*) есть соединение аппаратных компонентов (центральный процессор, процессоры ввода-вывода, память, периферийные устройства, интерфейсы) и программных компонентов (программы, составляющие операционную систему). Все эти компоненты часто называются *ресурсами* системы. Каждый компонент имеет собственные атрибуты, которые мы будем называть *параметрами* системы (даже если они не могут быть выражены отдельными числами; примером могут служить алгоритмы, по которым работает компонент). Так как предполагается, что читатель знаком с функциями и харак-

тористиками по крайней мере основных компонентов, мы не будем их описывать здесь. Следует, однако, заметить, что идентификация компонентов или ресурсов, входящих в систему, неоднозначна. Так как эта идентификация в значительной степени произвольна, она будет определяться соображениями удобства и поэтому может изменяться от исследования к исследованию даже для одной и той же системы.

Некоторые проблемы возникают при попытке точно определить *границы* вычислительной системы. Относительно включения в нее операционной системы сомнений возникать вроде бы не должно. Однако операционную систему можно определить как «набор ручных и автоматических процедур, которые позволяют группе людей эффективно использовать вычислительную установку» (Бринч Хансен, 1973). Таким образом, операторы (персонал), обычно выполняющие такие ручные операции, как пуск машины, установку лент и дисковых пакетов, укладку колод перфокарт в считывающее устройство, должны быть также включены в состав операционной системы. Присутствие людей в составе системы, подлежащей оценке, делает оценку более сложной и создает соблазн игнорировать их наличие. Впрочем, часто производительность системы настолько сильно зависит от расторопности ее операторов, что их присутствие никак нельзя игнорировать.

Другой тип проблем связан с математическим обеспечением общего пользования (компиляторы, редакторы, ассемблеры, программы отладки) и базами данных общего пользования: следует ли рассматривать эти программы и данные как часть вычислительной системы или как часть ее окружения? Наиболее разумный способ решения этих и других «граничных» проблем состоит в том, чтобы не принимать заранее всеобщих решений. Точное определение границ должно зависеть от частной задачи оценки, которую мы хотим решить, и, следовательно, остаться на усмотрение оценщика. Например, если оценивается производительность системы обслуживания базы данных, объект изучения будет включать не только комплекс аппаратура — операционная система, но и базы данных вместе с их обслуживающими программами. В исследованиях же, концентрирующихся вокруг производительности комплекса аппаратура — операционная система, те же самые программы и данные окажутся частью входного сигнала из внешнего мира.

Таким образом, мы оставим на время некоторые детали в пограничной области неопределенными с тем, чтобы охватить нашим методом более широкий класс задач. Однако не следует никогда забывать специфицировать очень точно границы системы, как только стали ясны цели исследования.

Термин *установка* будет использоваться по отношению к комплексу, состоящему из определенной системы и ее среды. Хотя *среда* системы и не определяется точно ввиду упомянутых выше граничных трудностей, она всегда включает *коллектив пользователей*. Это сообщество таких людей, которые используют систему для удовлетворения своих нужд по обработке информации. Существуют и другие люди, которых

следует рассматривать либо как часть среды системы, либо как часть самой системы (например, начальник установки и члены обслуживающей бригады, персонал по ремонту машины, администрация, отвечающая внутри организации за обработку информации). Так как в этой книге мы сосредоточимся на технических аспектах оценки производительности, наши интересы в большинстве случаев будут фокусироваться на среде, практически совпадающей с коллективом пользователей. Эти люди воздействуют на производительность системы, создавая входные сигналы (программы, данные, команды), которые в совокупности будут обозначаться термином *рабочая нагрузка*. Система в свою очередь будет влиять на свою среду, создавая поток выходных данных с определенной производительностью. Эта производительность является для среды частью ее входных данных и, следовательно, влияет на ее поведение (см. рис. 1.1).

Атрибуты рабочей нагрузки будем называть *параметрами рабочей нагрузки*, даже если они не могут быть выражены отдельными числами.

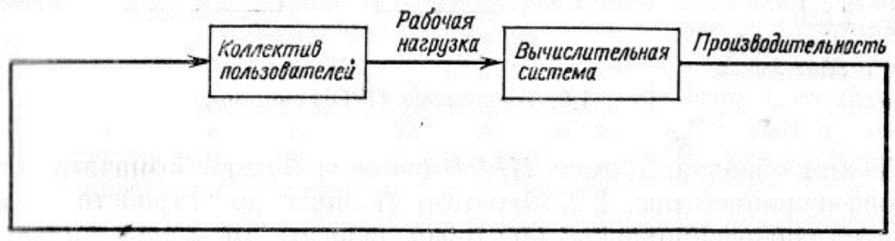


Рис. 1.1. Блок-схема вычислительной установки.

Параметры системы и параметры рабочей нагрузки вместе будут обозначаться термином *параметры установки*.

При любой модификации системы получается в принципе другая система. Однако, если модификация минимальна, мы часто будем говорить, что получилась новая *конфигурация* или новая *версия* той же системы. Термин «конфигурация» обычно относится к набору системных компонентов и к способу их соединения. Термин «версия», наоборот, ориентирован на программное обеспечение.

Из-за дороговизны работы по оценке производительности ожидаемая выгода от этой работы особенно ощутима тогда, когда ее объектом является или относительно большая и дорогая система или же дешевая система массового производства, которая будет продаваться в больших количествах. Особое внимание в этой книге будет обращено на системы среднего размера, но принципы и некоторые приемы, которые будут описаны, применимы также к меньшим и менее сложным системам, таким, как мини- и микрокомпьютеры.

Система с конфигурацией, изображенной на рис. 1.2, выдвигает ряд интересных и важных для практики проблем оценки, несмотря на то что она состоит из очень ограниченного количества аппаратных компонентов. В сущности сложность системы не так сильно зависит от конфигурации аппаратуры, как от способа организации ресурсов и

способа управления ими со стороны операционной системы. Наши примеры в этой книге будут ограничены очень простыми конфигурациями и схемами управления. Многие из этих примеров основываются на конфигурации рис. 1.2, которая для удобства будет называться ПРОП (конфигурация ПРимера для изучения Оценки Производительности).

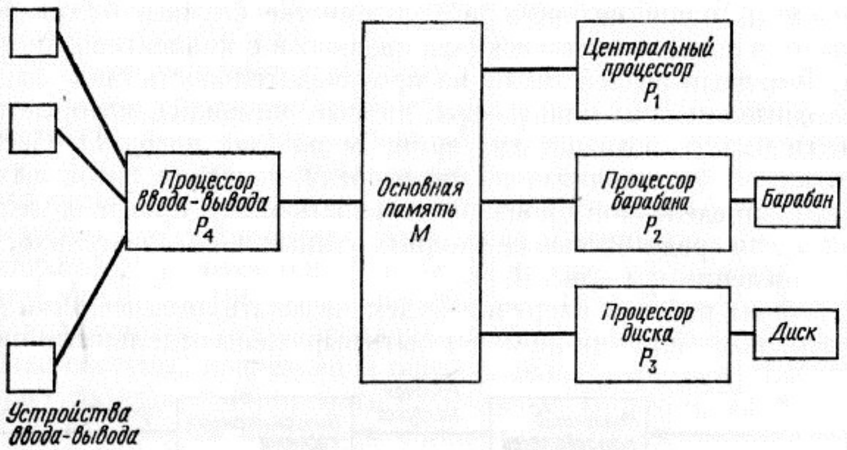


Рис. 1.2. Блок-схема ПРОП-системы.

Таким образом, термин *ПРОП-система* будет обозначать систему с конфигурацией рис. 1.2. Отметим, что природа устройств ввода-вывода не специфицирована; она будет зависеть от типа системы, рассматриваемой в каждом конкретном случае. Типовые структуры систем, с которыми мы будем иметь дело на протяжении этой книги, перечислены в табл. 1.1 и будут кратко описаны ниже применительно к конфигурации ПРОП.

Таблица 1.1

Основные типы организации рассматриваемых систем

Режим обработки
Пакетный
Диалоговый
Совмещение
Однопрограммные
Мультипрограммные
Управление памятью
Ручное
Автоматическое (виртуальная память)

Типовыми устройствами ввода-вывода *системы пакетной обработки* (в том числе и конфигурации ПРОП) являются устройства чтения перфокарт, карточные перфораторы и устройства печати. Пользователи предъявляют свои *задания* (программы и данные) пробитыми на картах. Карты читаются устройством чтения перфокарт, и после преобразования кодов их содержимое переносится в буферную область в основной памяти M через P_4. Когда буфер наполнится, его содержимое переносится на диск через P_3. На диске задания ждут своей очереди на загрузку в память. Как только задание загружено через P_3, оно может обрабатываться центральным процессором P_1. Во время выполнения задание будет, вообще говоря, требовать информацию, расположенную на барабане или на диске. Операционная система, расположенная в памяти M (хотя бы частично), будет заботиться об обменах. Задание также будет генерировать выходное сообщение для пользователя, которое накапливается на диске, а в конце выполнения задания печатается, проходя через P_3, M и P_4. Пользователи системы пакетной обработки сообщают операционной системе свои запросы ресурсов на *языке управления заданиями* посредством колоды карт задания. В одном задании, т. е. в одной и той же колоде, может быть затребовано последовательное выполнение нескольких шагов задания. *Шаг задания* есть этап выполнения задания, логически отличный от остальных и явно специфицированный предложением языка управления заданиями. Типичные шаги задания: компиляция, выполнение компилированной программы, копирование файла с диска на барабан или, наоборот, распечатка содержимого файла. Когда шаг задания оканчивается, отведенная ему область памяти освобождается, а следующий шаг того же задания добавляется к списку шагов, ожидающих загрузки в M.

Если все процессоры на рис. 1.2 в действительности представляют один и тот же физический процессор, скажем P*, то совмещение во времени невозможно. В то время когда P* действует как P_4, он не может действовать как P_1, P_2 или P_3. В этом случае задания выполняются строго последовательно; было бы неудобно начинать выполнение нового задания, прежде чем завершится предыдущее. Такую организацию, в которой не допускается совмещение, следовало бы назвать *чисто однопрограммной*.

Пусть теперь процессоры на рис. 1.2 представлены четырьмя независимыми процессорами. Процессоры P_2, P_3 и P_4 называются *каналами ввода-вывода* и могут представлять собой либо *контроллеры* (процессоры с постоянными программами), либо обычные процессоры с запоминаемыми программами. Деятельность этих четырех процессоров может перекрываться во времени. Тогда, например, пока происходит передача информации между устройством чтения перфокарт и диском или между диском и печатью, P_1 может выполнять задание. Этот тип системы будем называть СПУ (СПУ означает Совмещение работы Периферийных Устройств во взаимодействии с центральным процессором). В такой системе работа центрального процессора и канала ввода-вывода может перекрываться. Однако в памяти все время находится только одно выполняемое задание пользователя. Таким образом, несмотря на независимость процессоров, центральный процессор не может переключаться на другие задачи во время пересылок ввода-вывода, заказанных текущей задачей, и должен ждать их завершения. Итак, в системе СПУ совме-

щение ограничено; например, невозможно совместить работу P_1 и P_2, если выполняемое задание не может продолжать свои вычисления после выдачи запроса на барабан.

Если в память может быть загружено несколько заданий пользователя, центральный процессор может переключаться с одного задания на другие, так как выполняемое задание нуждается во вводе-выводе, а для того, чтобы произошла такая передача данных, операционная система должна совершить необходимые приготовления. Мы будем называть задание *активным*, если оно находится в данный момент в памяти. Переключение центрального процессора возможно, только если хотя бы одно из других активных заданий *готово* к работе. Активное задание не готово (или *блокировано*), если оно ожидает завершения заказанного им ввода-вывода. Такой тип систем мы будем называть *мультипрограммными системами*. В такой системе число активных заданий обычно называется *степенью мультипрограммирования*. Термин *однопрограммная работа* будет, вообще говоря, применяться также и к системам типа СПУ, несмотря на одновременное выполнение некоторых действий в таких системах, поскольку центральный процессор не переключается на другое задание, пока выполняемое задание блокирует само себя запросом на ввод-вывод.

Как в однопрограммных, так и в мультипрограммных системах программа, по-видимому, должна быть полностью загружена в память, чтобы началось ее выполнение. На каждое задание отводится определенный объем памяти. Если размер задания превосходит этот объем, программист должен разделить задание на *сегменты*, которые должны быть последовательно загружены операционной системой в соответствии с детальными указаниями пользователя. Эта организация называется (по очевидным причинам) *ручным управлением памятью*.

Однако возможно (и особенно удобно в мультипрограммных системах) построить механизмы, которые позволят программам выполняться даже будучи загруженными лишь частично, без жестких ограничений, налагаемых методом перекрытия сегментов, и без вмешательства программиста. Это может быть достигнуто применением схем *автоматического управления памятью*, которые позволяют операционной системе решать, какая часть программы должна находиться в памяти в данный момент. Ссылка выполняемой программы на раздел информации, не находящийся в основной памяти, вызывает *отказ*. Отказы обрабатываются операционной системой, которая считывает отсутствующий раздел с внешнего запоминающего устройства, выделенного для этой цели (например, с барабана). Таким образом, *иерархия памяти*, состоящая из М и барабана, автоматически управляется операционной системой, которая ведет обмен информацией между М и барабаном без всяких указаний со стороны программиста. Действительно, имея дело с такими системами, программисты пишут свои программы в пространстве адресов, не совпадающем с пространством физических адресов, т. е. с М. Это пространство называется *пространством виртуальных адресов*, или множеством адресов *виртуальной памяти*. Размер виртуальной памяти одного задания обычно много больше М (или области М, отведенной одному заданию). Таким образом, программисту только изредка приходится заботиться о приспособлении заданий к имеющемуся размеру памяти. Наложение виртуальных адресов на физические (в М или в барабане) выполняется динамически, ссылка за ссылкой, специальным аппаратным механизмом.

Типичными устройствами ввода-вывода диалоговой ПРОП-системы являются телетайпы или дисплеи. Пользователи вводят *команды* вместо предложений языка управления заданиями и *данные* для системных программ или программ пользователя, с которыми они взаимодействуют. Эти программы должны существовать в системе до того, как они будут названы командами пользователя. Каждое *сообщение*, введенное пользователем, состоящее из команды или данных, или из того и другого, собирается символ за символом процессором P_4 в буфере внутри М. Когда поступает символ «конец сообщения», операционная система интерпретирует сообщение и «активирует» процесс, с которым пользователь желает взаимодействовать. Обычно этот процесс содержит бóльшую часть своей информации на диске или на барабане. Активация процесса означает в данном случае постановку его в очередь процессов, ждущих загрузки в память. В некоторый момент информация для процесса (программы и данные) загружается полностью или частично в М, и как только процессор подключится к нему, начнется его выполнение.

Пользователь, вводящий «легкую» команду (требующую малых ресурсов системы), ожидает быстрого ответа системы. Например, команда редактирования строки должна быть выполнена самое большее за несколько секунд. Иначе внимание пользователя будет рассеиваться, и взаимодействие с системой станет тягостным и неэффективным.

Однако некоторые из команд, подлежащих обслуживанию в данный момент времени, вполне могут оказаться «тяжелыми»; например, некоторые пользователи могут заказать компиляцию, выполнение длинной программы или манипуляцию с большим файлом; тогда как остальные просто выполняют редактирование, спрашивают время дня или используют систему в качестве настольного калькулятора. Обычно, для того чтобы предотвратить воздействие тяжелых запросов на время ответа системы на легкие вопросы, используется метод разделения времени. Этот метод состоит в ограничении времени, на которое центральный процессор передается данному процессу без переключения на другие готовые процессы. Очевидно, что центральный процессор переключается, только если существует другой готовый процесс. Максимальное время центрального процессора, которое может быть дано некоторому процессу каждый раз при его включении, называется *квантом процессорного времени*. Если выполнение процесса не завершилось, когда истек квант, процесс прерывается и ставится в очередь готовых. Когда подходит его очередь, процесс получает следующий

квант и т. д. до его завершения. Заметим, что завершение в диалоговой системе обычно означает, что процесс требует новой команды или новых данных от пользователя.

Диалоговая система может быть однопрограммной или мультипрограммной. В первом случае только один процесс в данный момент времени находится в основной памяти; когда он оканчивается или его время превосходит квант, процесс *разгружается* из памяти и в память *загружается* другой процесс, готовый к выполнению. В случае мультипрограммирования в памяти одновременно могут находиться несколько процессов, и центральный процессор переключается на готовый процесс, если текущий процесс блокирует сам себя операцией ввода-вывода. Если это терминальная операция ввода-вывода (т. е. процесс ею заканчивается) или квант исчерпан, процесс обычно (но не всегда) разгружается из памяти.

Иерархия памяти даже в диалоговой системе может управляться вручную или, по крайней мере частично, автоматически. В последнем случае для того, чтобы процессы могли выполняться, не обязательно, чтобы они целиком загружались в память или были должным образом разделены на сегменты. Иначе говоря, диалоговая система (однопрограммная или мультипрограммная) может предоставить каждому из своих пользователей виртуальную память.

Читатель должен бы отметить, что три организационных координаты, отмеченные в табл. 1.1 не зависимы друг от друга; любая из восьми комбинаций направлений (по трем координатам) принципиально возможна. В некоторых системах оба варианта, указанные в табл. 1.1 для определенной координаты, сосуществуют. Например, имеется много операционных систем, обеспечивающих пакетную обработку наряду с диалоговой. Возможно также автоматическое управление частью иерархии памяти одновременно с ручным управлением остальной частью.

Важно отметить, что мультипрограммирование было введено в основном для увеличения количества работы, выполняемой центральным процессором,— следовательно, для увеличения производительности системы,— и его наличие скрыто от пользователя. С другой стороны, метод разделения времени обосновывался соображениями скорости ответа на команды пользователя. Более того, пользователь видит макроскопическую разницу между пакетной обработкой и диалоговым счетом. Наличие виртуальной памяти также воздействует на пользователя, но не так глубоко и непосредственно, как способ обработки.

1.3. ЭТАПЫ РАБОТЫ ПО ОЦЕНКЕ

Полезно различать *работы по оценке производительности и непрерывное наблюдение за состоянием* системы. Хотя оба вида деятельности ограничены по длительности, так как время существования системы конечно, непрерывное наблюдение за состоянием обычно выполняется для значительной части времени существования работающей системы.

Его целью является постоянное наблюдение за производительностью системы, чтобы обнаружить проблемы производительности, как только они возникнут.

Работа по оценке производительности гораздо сильнее ограничена во времени, не всегда требует существования системы и обычно начинается с идентификации некоторой проблемы производительности или хотя бы с предположения о ее существовании. В этой книге мы

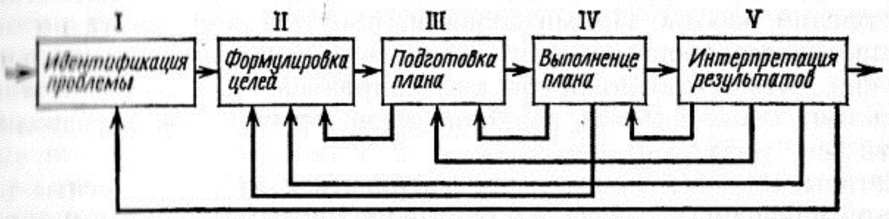


Рис. 1.3. Этапы работы по оценке производительности.
Этап I. Возникает необходимость проведения работы.
Этап II. Формулируются цели работы.
Этап III. Подготавливается план работы.
Этап IV. План выполняется.
Этап V. Интерпретируются результаты.

сосредоточимся на исследованиях с целью оценки, хотя методы и средства измерения, описанные в гл. 2, включают в себя также те, что применяются для непрерывного наблюдения за состоянием системы.

Виды деятельности, составляющие вместе работу по оценке, могут быть сгруппированы в пять этапов, наглядно представленных на блок-схеме рис. 1.3.

Рисунок 1.3 показывает также некоторые из возможных путей, которые могут быть пройдены в ходе работы по оценке.

Необходимо сделать несколько общих замечаний по перечисленным выше этапам работы. Прежде всего, этапы II и III должны включать в себя тщательную оценку всех входящих затрат и возможных выгод. Каждое решение относительно целей и ресурсов, которые придется затратить в ходе исследования, должно принимать во внимание критерий затраты — выгода. Тщательно спланированные работы по оценке часто окупаются, но для этого они должны планироваться весьма тщательно. Предсказывать с некоторой степенью надежности выгоды, которые могут быть получены от работ по оценке, обычно очень трудно, гораздо труднее, чем предсказывать затраты. На вопросы об усовершенствованиях, которые могут быть получены в результате некоторого исследования, например, относительно легко отвечать в конце исследования; в начале, не зная результатов работы, ответить на них надежно почти невозможно. Таким образом, оценщики вынуждены будут полагаться на свой опыт в подобных случаях, на оценки наилучшего и наихудшего случаев, кустарные приемы и свое везение. Однако долж-

ны предприниматься и серьезные попытки, подразумевающие учет соотношений затраты — выгода.

Другое, возможно очевидное, но существенное замечание состоит в том, что важно как можно яснее определить область исследования и его цели прежде, чем будет сделано что-либо еще. Следовательно, эта спецификация должна быть сделана на этапе II. Эти цели и, возможно, цели, возникающие в ходе исследования, должны служить ориентиром для производящих оценку на протяжении всех этапов и повторений работы. Иными словами, практическая работа по оценке (в противовес теоретическому проекту оценки производительности) должна быть по возможности целенаправленной. Во время этапа II предстоит также выбрать подходящий набор индексов производительности (см. разд. 1.4).

Этапы III, IV, V могут рассматриваться как компоненты итеративной процедуры, лежащей в основе научного метода: формулируется некоторая гипотеза, затем она проверяется, и если результат отрицательный, то модифицируется. В оценке производительности, как будет видно из дальнейшего, гипотезы не всегда проверяются экспериментально. Однако экспериментальное подтверждение остается важнейшим критерием истинности. Следует также заметить, что оценщики могут формулировать гипотезы с гораздо большими шансами на успех, если они знают внутреннюю организацию и поведение системы. Не существует реальной замены знанию, а подход «черного ящика» не всегда так уж эффективен по стоимости. Чем глубже понимание системы, тем, вообще говоря, более дешевым и быстрым окажется исследование. Иногда специалист по системе может безошибочно ответить на такие вопросы, которые потребовали бы от других людей продолжительных и дорогих исследований.

Читателю следует также иметь в виду, что каждый из перечисленных выше этапов включает в себя деятельность, направленную на накапливание информации о системе (и данной установке): чтение руководств по системе, изучение логических диаграмм, блок-схем, листингов программ, расспросы операторов и другого обслуживающего персонала, беседы с пользователями, измерение параметров рабочей нагрузки, сбор данных об использовании системы, интерпретация учетных данных и т. д.

1.4. ИНДЕКСЫ ПРОИЗВОДИТЕЛЬНОСТИ

В разд. 1.1 была обрисована аналогия между технической концепцией производительности и экономической концепцией ценности. Было также сделано замечание, что, подобно ценности, концепция производительности субъективна. Это означает, что разные люди стремятся использовать различные индексы производительности при оценке систем. *Индекс производительности* есть описатель, который используется для представления всей производительности системы или какого-то из ее аспектов.

Если мы попросим несколько людей упорядочить различные вычислительные системы в общем диапазоне цен, вероятно, что полученные номера будут отличаться. В оценке различных альтернатив эти люди будут пользоваться своими индивидуальными субъективными индексами производительности. Порядковый номер, предложенный каждым из людей для каждой системы, будет некоторым образом выражать ценность, приписываемую данным человеком данной системе.

Так как производительность в соответствии с нашим широким определением в разд. 1.1 эквивалентна ценности, она может быть измерена в долларах (или других денежных единицах) и, следовательно, может быть сведена к всеобщей количественной основе оценки. Однако это не только не делает производительность более объективной концепцией, но и не дает инженерам по вычислительной технике советов, как сконструировать лучшие системы или улучшить существующие. Вместо этого мы можем попытаться перевести различные субъективные представления о производительности в термины более объективных, возможно допускающих количественную форму, технических индексов производительности. Эти индексы, если они аккуратно определены, могли бы быть объективными мерами некоторых аспектов производительности системы. Таким образом, субъективность оценки могла бы быть в значительной степени ограничена их выбором и относительным весом, придаваемым им. Веса будут, вообще говоря, отличаться с точки зрения разных категорий людей (изготовителей, администрации установки, представителей компании, отраслевых инженеров, системных программистов, операторов, прикладных программистов) и иногда от индивидуума к индивидууму.

Однако ряду важных индексов производительности трудно или невозможно придать количественный вид, например *легкость пользования* системой, *структурность* программы или языка, *мощность* набора команд. В этой книге мы примем ограниченное определение производительности, которое предлагает индексы, легко принимающие количественную форму. Только эффективность системы, ее скорость выполнения данной задачи или набора задач, быстрота ответа на внешние сигналы будут рассматриваться из аспектов широкого определения производительности, данного в разд. 1.1. Таким образом, мы зарезервируем термин *производительность* для этих аспектов и будем называть остальные (правильность, легкость использования, структурность, надежность и т. д.) *функциональными аспектами*.

Наиболее широко распространенные классы количественных индексов производительности для вычислительных систем перечислены в табл. 1.2. Из общих определений, данных в той же таблице, должно быть очевидно, что *индексы продуктивности* имеют размерность: объем×время$^{-1}$, *индексы реактивности* — размерность времени, а *индексы использования* безразмерны. В настоящее время не существует стандартизированного единого способа измерения *объема*, или *количества информации*, переработанной системой (некоторые попытки получения универсальной меры будут упомянуты в разд. 5.2.3). Таким

Таблица 1.2

Основные классы количественных индексов производительности вычислительных систем

Класс индекса	Примеры индексов	Общее определение
Продуктивность	Пропускная способность Скорость выработки Максимальная выработка (максимум пропускной способности) Скорость выполнения команд Скорость обработки данных	Объем информации, обрабатываемой системой в единицу времени
Реактивность	Время ответа Время прохождения Время реакции	Время между предъявлением системе входных данных и появлением соответствующей выходной информации
Использование	Коэффициенты использования оборудования (центральный процессор, канал ввода-вывода, устройство ввода-вывода) Коэффициент использования операционной системы Коэффициент использования общего модуля программного обеспечения (например, компилятора) Коэффициент использования базы данных	Отношение времени использования указанной части системы (или ее использования для заданной цели) в течение заданного интервала времени к длительности этого интервала

образом, в зависимости от системы и от ее рабочей нагрузки будут использоваться различные меры объема; среди наиболее распространенных можно назвать: *задание*, *программу*, *процесс*, *шаг задания*, *задачу*, *сообщение*, *взаимодействие (обмен сообщениями)*, *команду*. Перечислить все значения, приписанные ранее и приписываемые ныне этим терминам в литературе по вычислительным системам, по-видимому, невозможно. Здесь мы только отметим, что все они до некоторой степени зависят от природы рабочей нагрузки, от языка, на котором программисты описывают свои алгоритмы для машины, от внутреннего языка машины и от способа организации системы. Таким образом, ни одна из этих мер не обладает свойством независимости от рабочей нагрузки и свойством независимости от системы — это два свойства, необходимые для того, чтобы можно было установить некоторую меру объема информации в качестве универсальной. Мы отложим точное определение значений, которые мы припишем большинству упомянутых выше терминов, а также и некоторым из терминов во втором столбце табл. 1.2, до первого их использования. Оценщики всегда должны однозначно определить, что означает определенная мера объема в конкретном контексте их работы.

В следующем примере мы обсудим выбор индексов производительности для диалоговой системы.

Пример 1.1. Диаграмма на рис. 1.4 характеризует диалоговую установку с точки зрения отдельного пользователя, взаимодействующего с системой через терминал. Заметим, что часть цикла взаимодействия, в течение которой машина ожидает сообщения от терминала, часто называется *временем обдумывания*, даже если она

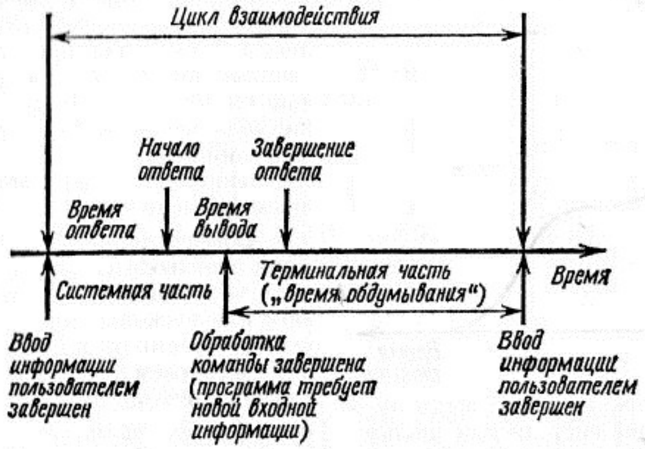


Рис. 1.4. Определение времени взаимодействия и времени ответа для диалоговой системы.

включает также время ввода и часть времени вывода. Для пользователя наиболее важным аспектом производительности (в соответствии с нашим ограниченным определением этого термина) будет реактивность системы. Диаграмма на рис. 1.4 позволяет нам точно определить индекс реактивности, называемый *временем ответа*. Например, время ответа будет определено как интервал между моментом, когда пользователь набирает последний символ входного сообщения, и моментом появления первого символа выходного сообщения на терминале. Это только одно из нескольких возможных определений, которые можно было бы предложить. В некоторых системах оно может оказаться даже двусмысленным и нуждается в модификации, по крайней мере в уточнении.

Важно отметить, что соотношение между индексами производительности системы и ценностью (в экономическом смысле), приписанной им людьми, участвующими в оценке, не обязательно является линейным. Например, никак нельзя считать верным, что время ответа 4 секунды на данное сообщение приблизительно вдвое менее желательно, чем время ответа 2 секунды на то же сообщение. Форма этой зависимости, которая меняется от пользователя к пользователю и от изменения типа сообщения, скорее похожа на кривую, изображенную на рис. 1.5. Ниже определенного порога пользователи не в состоянии даже ощущать разницу во временах ответа. Выше другого порога уровень удовлетворения резко падает, и преобладает ощущение разочарования и бессилия. Разработчикам систем и оценщикам всегда следует уделять большую долю внимания характеристикам пользователей.

Какие параметры установки влияют на время ответа? Вообще говоря, время ответа зависит от данного сообщения, от системы и от остальной части рабочей нагрузки, которую обрабатывает система во время приема сообщения. Чтобы пользоваться временем ответа в качестве индекса производительности системы, мы должны специфицировать сообщение и остальную часть рабочей нагрузки. Его значение бессмысленно без этой дополнительной информации. Однако охарактеризовать рабочую нагрузку в достаточно компактном виде весьма трудно, как мы увидим на протяжении этой книги и особенно в гл. 5. Более того, сообщения, представляющие интерес для

коллектива пользователей и даже отдельного пользователя, многочисленны и сильно различаются друг от друга по запросам ресурсов. Следовательно, мы будем рассматривать время ответа как случайную переменную, и в качестве индекса производительности нашей системы рассмотрим его распределение за данный период времени. Если этот период достаточно велик, маловероятно, что особенности отдельных сообщений и различных условий нагрузки, существующих в момент обработки сообщений, повлияют заметным образом на статистическое распределение времен ответа. Это распределение затем может быть использовано для того, чтобы представлять поведение системы под рабочей нагрузкой, которую она обрабатывает в течение выделенного периода времени. Обоснование принятых решений надо очень ясно представлять себе, занимаясь измерением или оцениванием времени ответа системы.

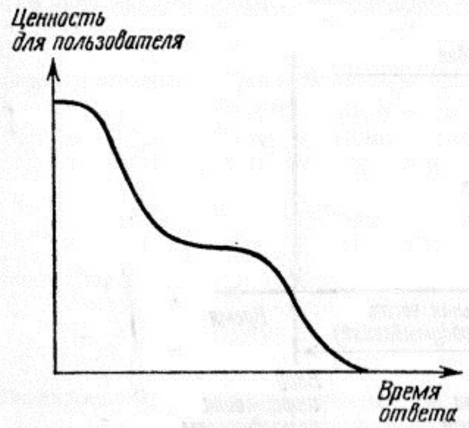


Рис. 1.5. Возможная зависимость ценности диалоговой системы для пользователя от времени ответа.

Обычно желательно работать с индексами производительности, значения которых могут быть представлены одним числом или несколькими числами. Таким образом, вместо распределения времени ответа мы можем использовать в качестве индекса производительности его первый момент (*среднее время ответа* $\bar{t}$) или его два первых момента (среднее $\bar{t}$ и стандартное отклонение σ_t).

Рассмотрим диалоговую ПРОП-систему с шестью активными пользователями, которые приблизительно одновременно вводят шесть команд. Эта система мультипрограммная, но без виртуальной памяти. Предположим, что программы, которые будут выполняться, полностью загружены в память к моменту поступления команд, и ни одна из команд не выдает запроса на диск или барабан. Во время обработки упомянутых команд никакие дополнительные команды не поступают. Из этих команд четыре принадлежат к типу редактирующих (предположим, что каждая из них требует для своего выполнения одну единицу времени центрального процессора), а остальные две — команды компиляции (каждая требует 4 единицы времени центрального процессора). Порядок выполнения команд совпадает с порядком их поступления: сначала одна команда редактирования (c_1), затем две команды компиляции (c_2, c_3), за которыми в свою очередь следуют остальные три команды редактирования (c_4, c_5, c_6). Когда время использования центрального процессора некоторым процессом превосходит квант, равный 2 единицам времени, процесс прерывается и ставится в очередь готовых процессов. Для выполнения из очереди выбирается следующий процесс по правилу: «первым пришел — первым обслужен». Переключение центрального процессора с одного процесса на другой требует 0.2 единицы времени центрального процессора. Это единственный тип *накладных расходов* (т. е. потребление ресурсов, не связанное прямо ни с одним процессом пользователя), который мы принимаем во внимание в этом крайне простом примере.

Обработка этих шести команд графически представлена на временной диаграмме [рис. 1.6 (а)]. Моменты завершения процессов на оси времени обведены кружками. Так как все команды введены в начальный момент времени, те же числа являются также временами ответа в предположении, что выходные сообщения печатаются в самом конце каждого процесса. Среднее время ответа $\bar{t}$ равно 8.16 единицы времени при стандартном отклонении σ_t, равном 9.55 единицы.

Очевидно, что пользователи, занятые редактированием, более заинтересованы в реактивности системы на команды редактирования, а пользователи, проводящие компиляцию,— в реактивности на команды компиляции. Таким образом, мы можем взять в качестве индексов производительности распределения времен редактирования и компиляции или соответствующие их моменты. В случае, представленном

на рис. 1.6 (а), среднее время ответа на команды редактирования составляет 6.1 единицы времени, а на команды компиляции 12.3 единицы.

Точка зрения пользователя не может быть полностью принята администратором установки, который должен беспокоиться о продуктивности системы. Ясно, что минимальная стоимость среднего времени ответа будет достигнута, если каждая команда будет выполняться сразу же при поступлении. Это было бы так, если бы система была полностью отдана в распоряжение одного пользователя. В системе с многими пользователями этого можно добиться, только если система всегда поддерживается очень

Рис. 1.6. Временные диаграммы выполнения команд $c_1,\ldots,c_6$ из примера 1.1 : (а) — квант времени: 2 единицы; (b) — квант времени: 1 единица. Моменты завершения выполнения команд обведены кружками. Команды редактирования c_1, c_4, c_5, c_6 требуют по одной единице времени, а команды компиляции c_2, c_3 требуют по четыре единицы времени каждая.

слабо загруженной. Однако центральный процессор будет простаивать большую часть времени ожидания сообщения от каждого терминала, участвующего в диалоге (см. рис. 1.4). Таким образом, его производительность, которая может быть выражена числом взаимодействий, завершившихся в единицу времени, окажется очень низкой. На рис. 1.6 (а) у пользователей, которые ввели команды, завершившиеся первыми, времена обдумывания частично перекрываются с выполнением других команд.

Администратор установки будет, следовательно, заинтересован не только в качестве обслуживания пользователей, но и в *пропускной способности*, определенной как число взаимодействий, обработанных системой в единицу времени. Это отношение, вообще говоря, является функцией времени. Оно может быть сведено к одному числу путем рассмотрения его среднего по времени (*средняя пропускная способность* $\bar{T}$) в качестве первичного индекса производительности. На рис. 1.6 (а) шесть взаимодействий завершены за 13.4 единицы времени. В этот период не входят интервалы ожидания сообщений от терминалов, но входят действия с терминалами, относящиеся к последующим взаимодействиям. Эти «краевые эффекты» важны здесь из-за очень короткой длительности периода. Применяя введенное выше определение средней пропускной способности, получим $\bar{T}=6/13.4=0.447$ взаимодействия в единицу времени.

Различные индексы производительности описывают разные аспекты поведения системы в рамках одной установки. Следовательно, они, вообще говоря, не независимы друг от друга, а скорее связаны неко-

торым образом. Улучшение одного индекса не всегда улучшает другие индексы, как показано в следующем примере.

Пример 1.2. В диалоговой ПРОП-системе, рассмотренной в примере 1.1, уменьшим квант времени с 2 единиц времени до 1 единицы. Новая временная диаграмма изображена на рис. 1.6 (b). Используя те же самые индексы производительности, что и в примере 1.1, найдем, что среднее время ответа уменьшается с 8.16 до 7.6 единицы времени. Однако средняя пропускная способность также падает с 0.447 до 0.422 взаимодействия в единицу времени. Заметим, что стандартное отклонение времени ответа увеличивается с 9.55 до 11.35 единицы времени. Среднее время ответа для команд редактирования убывает с 6.1 до 4.6 единицы времени, тогда как для команд компиляции оно увеличивается с 12.3 до 13.6 единицы времени.

■

В работе по оценке можно рассматривать *вторичные* индексы производительности наряду с *первичными*. Первичные индексы — это те, которыми действительно интересуется оценщик. Спецификации производительности системы обычно выражаются в терминах первичных индексов. Вообще целью работы по оценке на существующей системе является улучшение первичных индексов. Эти индексы меняются в зависимости от типа системы, задачи и от оценщика. Например, для пользователей обычно первичный индекс — это индекс реактивности. С другой стороны, администрация установки обычно заинтересована в продуктивности наряду с реактивностью. В зависимости от обстоятельств администратор будет уделять больше внимания то одному, то другому показателю, но оба типа индексов всегда будут играть главную роль в оценке системы администратором. Таким образом, даже в тех работах, цели которых формулируются в терминах одного типа индексов, другому типу всегда следует уделять должное внимание.

Вторичные индексы обычно выдвигаются в процессе самой работы по оценке. Их значения могут быть полезны для выявления симптомов неэффективности или для нахождения причин этой неэффективности. В качестве примера для ряда таких работ можно назвать индексы типа коэффициента использования (см. табл. 1.2).

В большинстве работ значения первичных и вторичных индексов производительности приходится определять для данных значений параметров установки. Эта операция может быть названа *анализом производительности*. Противоположную ей операцию, определение параметров по данным значениям индексов, назовем *синтезом производительности*. Если бы существовали подходящие методы синтеза, они нашли бы естественное применение в проектировании систем. Однако поскольку это не так, анализ производительности также широко применяется и при проектировании, как мы увидим в гл. 8. Методы оценки, которые будут введены в разд. 1.6, все по существу принадлежат к методам анализа производительности. Этот анализ производительности может быть повторен для ряда различных комбинаций значений параметров установки. Посредством такого метода мы можем исследовать пространство параметров, т. е. собрать информацию о зависимости индексов производительности от параметров и друг от друга. Такое исследование важно для нашего понимания поведения системы.

1.5. КЛАССИФИКАЦИЯ РАБОТ ПО ОЦЕНКЕ

Цели работы по оценке должны быть ясно и четко сформулированы в ее начале на этапе, который был назван этапом II. Работы по оценке могут быть классифицированы по различным координатам в соответствии с их целями. Возможно, наиболее распространенной является классификация, в соответствии с которой работы делятся на следующие три категории:

a) работы по выбору,
b) работы по усовершенствованию,
c) работы по проектированию.

Эта классификация, несмотря на расплывчатость границ между предлагаемыми ею классами, будет принята в книге. Мы введем эти классы работ (исследований) в оставшейся части данного раздела и обсудим их гораздо более подробно в гл. 6, 7 и 8.

Работы по выбору связаны с проблемами, встречающимися при проектировании установки и при покупке системы. Эти проблемы включают выбор способа обработки, выбор системы, которая будет установлена, из вариантов, предложенных различными поставщиками, выбор сервиса вычислительного обслуживания, выбор языка для написания определенной программы и выбор пакета системных программ. В целом работы по выбору могут быть определены как работы, предпринимаемые для определения наиболее подходящей альтернативы для данного применения среди различных доступных вариантов (например, из имеющихся в продаже). Работа по выбору проводится для достижения определенных целей посредством выбора существующей системы, которая наиболее соответствует некоторым точно определенным критериям выбора.

Работы по усовершенствованию относятся к модификациям существующих систем для улучшения их производительности или снижения цены, или для того и другого. *Настройка* системы, т. е. подбор параметров с целью приспособить ее к рабочей нагрузке установки, попадает в эту категорию работ. Сюда входят также работы, приводящие к *развитию* системы, которое состоит в замене или добавлении одного или более аппаратных компонентов. Например, система может развиваться путем расширения основной памяти, добавления центрального процессора или замены центрального процессора на более быстрый. К другим типам изменений, которые могут рассматриваться как модификации (типа настройки), относятся переупорядочение информации внутри одного из накопителей или среди нескольких накопителей, и модификация связей между устройствами и процессорами ввода-вывода. Работы этого класса могут быть названы *оптимизационными исследованиями*, если не придавать термину *оптимизация* строгого математического значения, которое в данном случае окажется дезориентирующим.

Цель *работ по проектированию* состоит в ответе на вопросы, возникающие при разработке вычислительных машин, их компонентов, операционных систем, программ и языков. Эти исследования обычно предпринимаются скорее изготовителями системы, нежели ее пользователями. Части, из которых будет составлена система, могут существовать заранее или нет. В последнем случае они будут спроектированы и изготовлены в соответствии со спецификациями, диктуемыми разработчиком системы. При разработке вычислительной системы ее применения часто определяются более широко и расплывчато, чем в работе по выбору. Например, система может быть создана для коммерческих применений (без дальнейшего уточнения этого понятия). Однако в процессе выбора эта система будет подвергнута испытанию, основанному на знании (возможно, расплывчатом, но много более точном, чем обладал разработчик) специфических коммерческих применений установки, для которых и проводится выбор.

Читателю следовало бы заметить, что данная задача оценки может классифицироваться по-разному в зависимости от контекста, в котором решается данная проблема. Это и есть аспект размытости границ классов, упомянутый выше. Например, определение удовлетворительного размера основной памяти системы может рассматриваться как задача выбора, если сопоставляемые альтернативные системы отличаются друг от друга по размеру памяти; это задача усовершенствования, если объем памяти окажется фактором, ограничивающим производительность существующей и работающей системы; это задача проектирования, если разработчики операционной системы желают определить минимальный размер памяти, необходимый для обработки реальной рабочей нагрузки с разумным уровнем производительности.

Однако следует также заметить, что классификации — всего лишь удобные способы организации определенной суммы знаний в мысленные структуры. Цель этих структур обычно состоит в подчеркивании макроскопических различий между классами, чтобы сделать дисциплину легче для понимания, изучения и запоминания. Таким образом, даже если мы найдем, что классификация работ по оценке, описанная в этом разделе, не вполне удовлетворительна, мы ее примем, поскольку находим ее разумно обширной и отвечающей нашим целям.

1.6. ОБЗОР МЕТОДОВ ОЦЕНКИ

Работа по оценке всегда направлена на получение ответов на вопросы о производительности данной системы. Чтобы сделать это, оценщик должен получить информацию о производительности. Обыкновенно эта информация состоит из значений индексов производительности системы при заданной рабочей нагрузке и при определенных значениях параметров системы. Иными словами, это информация, полученная в результате анализа производительности. Методы, посредством которых может быть получена эта информация, будут называться

методами оценки. Известные методы могут быть классифицированы разными способами. Самая распространенная классификация, которая и будет принята в этой книге, вводится ниже.

Информация о производительности, необходимая для исследования, может быть получена как от самой системы (*методы измерения*), так и от модели системы (*методы моделирования*).

Модель системы — это такое ее представление, которое состоит из определенного количества организованной информации о ней и построено с целью ее изучения. Поскольку существует очень много вопросов о системе, которые разумно задать, может быть сконструирован ряд различных моделей. Все эти модели представляют одну и ту же систему, но либо рассматривают ее с различных точек зрения и имеют различные цели, либо имеют различную степень детальности.

Модель может рассматриваться как система (заметьте, что здесь *система* не есть сокращение *вычислительной системы*). Различные модели некоторой системы соответствуют различным способам деления ее на компоненты и описания взаимодействия между компонентами или со средой системы. Возможность и удобство представления системы многими различными способами не должны нас удивлять; ведь наше знание мира основано на мысленных моделях «систем» вокруг нас (которые включают других людей и нас самих), а научный прогресс может рассматриваться как создание новых, лучших моделей мира. Описания ПРОП-систем, данные в разд. 1.2, могут рассматриваться как словесные и до некоторой степени поверхностные модели ряда реальных систем. Мы будем называть эти модели, которые существуют в умах оценщиков, *концептуальными моделями*.

Концептуальные модели вычислительных систем играют фундаментальную роль в оценке их производительности. Они состоят из всей информации, которой располагает оценщик об их структуре и поведении. Эта информация необходима, или по крайней мере очень полезна на всех этапах работы по оценке. Чем глубже наше знание системы, т. е. чем лучше наша концептуальная модель этой системы, тем легче и успешнее может оказаться исследование. Специалисты по системе часто могут ответить на такие вопросы по оценке производительности, которые потребовали бы от кого-нибудь другого продолжительной и дорогостоящей работы; таким образом, иногда удобная концептуальная модель — все, что нужно для работы. Итак, как будет показано в гл. 2, 3, 4 и 5, концептуальные модели являются основой методов измерения и двух типов методов моделирования: *имитации* и *аналитических методов*, представленных в этой книге.

Очень распространенное и удобное описание поведения системы (не только вычислительной системы) основывается на концепциях *состояния* и *перехода между состояниями*. Состояние системы в момент t определяется как множество значений интересующих нас параметров системы в момент t. Любое изменение этих значений может рассматриваться как переход к другому состоянию. Очевидно, что для любой данной системы существует бесконечное множество таких описаний.

Это описание, по-видимому, зависит от выбора интересующих нас параметров системы.

Если поведение модели во времени в основном воспроизводит поведение системы согласно некоторым условиям соответствия между различными аспектами модели и системы (в частности, между состояниями и переходами), мы имеем *имитационную модель*. Имитационная модель работает «точно так же», как сама система; наблюдается ее поведение во времени под внешним воздействием, которое представляет среду системы, и измеряются индексы производительности. Таким образом, существует концептуальное подобие между имитацией и измерением. Важным следствием этого подобия является то, что проблемы, возникающие при планировании измерительных экспериментов, идентичны или весьма похожи на проблемы, которые приходится решать для имитационных экспериментов. Решения, справедливые для одного типа эксперимента, часто применимы к другому (см. гл. 2 и 3).

С более математической точки зрения имитационная модель может рассматриваться как состоящая из уравнений, которые решаются путем прослеживания эволюции их решений на некотором отрезке времени. Вычислительные системы обычно рассматриваются как *дискретные* системы (с дискретными состояниями), которые изменяют свои состояния посредством предписанных дискретных переходов, называемых *событиями*. Таким образом, их имитационные модели будут включать уравнения, выражающие логические условия, при которых возникают эти переходы. Решение уравнений методом имитационного моделирования означает определение хронологической последовательности интересующих нас событий, возникающих в данной системе, и определение соответствующей последовательности состояний системы. Этот метод решения является, очевидно, численным.

Когда решение уравнений, составляющих модель, получено математическими методами, мы говорим, что использован *аналитический метод* и сконструирована *аналитическая модель*. Термин *аналитический* здесь несколько дезориентирует, так как по нашей классификации категория аналитических методов решения также включает все численные методы, кроме имитации. Однако за отсутствием более подходящего термина будем следовать традиции и использовать термин «аналитический» для обозначения этих моделей и методов.

Пример 1.3. В примере 1.1 мы использовали крайне простую имитационную модель, полученную из концептуальной модели диалоговой ПРОП-системы, рассмотренной там. Значения наших двух индексов производительности $\bar{t}$ и $\bar{T}$ вычислялись посредством моделирования (карандашом на бумаге) обработки шестикомандной рабочей нагрузки (см. рис. 1.6). Имитационная модель, показанная в виде блок-схемы на рис. 1.7, концептуально может трактоваться подобно системе, которую она, по-видимому, заменяет для целей нашего исследования. Это означает, что если мы снабдим эту модель подходящим представлением рабочей нагрузки, ответы модели будут воспроизводить ответы реальной системы. Подходящая модель рабочей нагрузки здесь состоит из последовательности процессов, каждый из которых характеризуется временем поступления и запросом на процессорное время. Заметим, что воспроизведение поведения системы будет зависеть от точности моделей системы и рабочей нагрузки.

Аналитическая модель этой же системы состоит из набора математических соотношений, которые могут быть использованы для вычислений (любым методом, кроме имитации) значений наших индексов производительности по заданным значениям параметров системы и рабочей нагрузки. Эта модель, подобно имитационной модели, рассмотренной выше, может заменить систему для целей нашего изучения. Например,

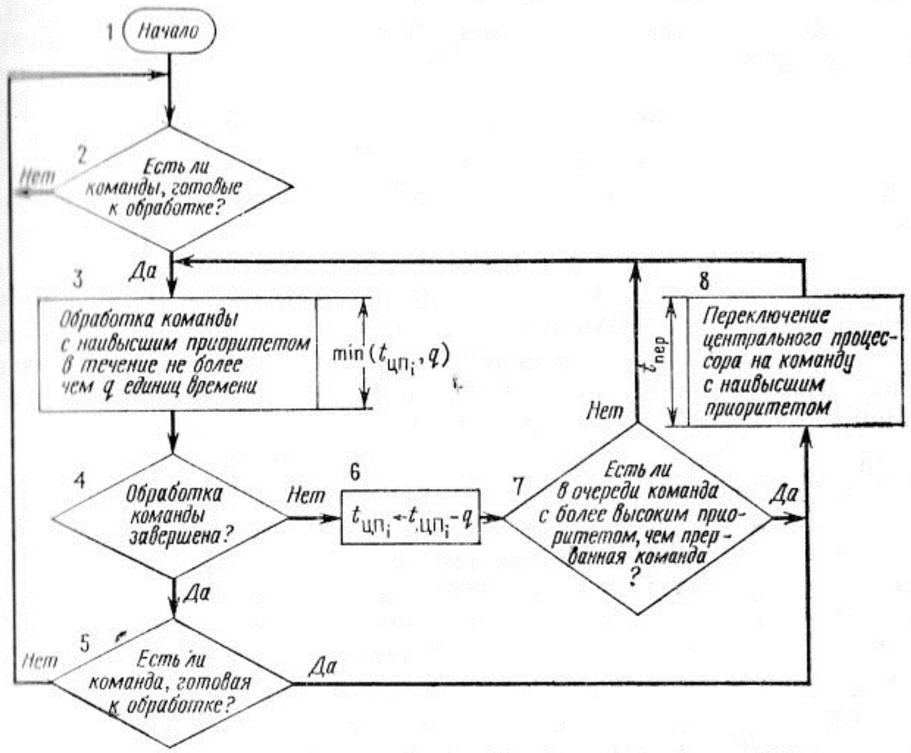


Рис. 1.7. Представление имитационной модели в виде блок-схемы, используемое для построения временной диаграммы на рис. 1.6. Единственными блоками, в которых затрачивается заметное время, являются блок 3[$t=\min(t_{цп}, q)$] и блок 8 ($t=t_{пер}$).

следующее простое выражение для средней пропускной способности, справедливое в предположениях примера 1.1, может быть получено непосредственно из рис. 1.6:

$$\bar{T} = \frac{n}{\sum_{i=1}^{n} t_{цп_i} + n_{пер} t_{пер}}, \tag{1.1}$$

где n — число команд, подлежащих обработке, $t_{цп_i}$ — время центрального процессора, необходимое на обработку i-й команды, $t_{пер}$ — время переключения центрального процессора с одного процесса на другой, а $n_{пер}$ — число переключений центрального процессора во время обработки рассматриваемых n команд. Таким образом,

$$n_{пер} = \sum_{i=1}^{n} \left\lfloor \frac{t_{цп_i}}{q} \right\rfloor - 1, \tag{1.2}$$

где q — квант времени, а $\lfloor x \rfloor$ означает наименьшее целое, большее или равное x. Написать уравнения, позволяющие вычислить среднее время ответа, гораздо труднее. В гл. 4 мы увидим, как это может быть сделано при более общих и реалистических предположениях. ■

Общей проблемой всех методов моделирования является проблема точности, т. е. соответствия моделируемой системе. Любая аналитическая или имитационная модель должна быть проверена, прежде чем ее можно будет использовать для получения информации, нужной для оценки. Проверка модели часто трудна, а иногда и невозможна. Она может опираться на предыдущие теоретические результаты или результаты моделирования, но если моделируемая система существует, окончательное обоснование процедуры выверки должно быть эмпирическим. Наше доверие к модели может исходить из нашей уверенности в ее разумной концептуальной основе и из использования корректной процедуры для ее построения. Однако нет лучшего способа (возможно, никакого другого способа) подтверждения нашей уверенности, чем эксперимент. Таким образом, в некотором смысле, измерение есть наиболее важный метод оценки, так как в нем нуждаются другие методы. Однако он не может быть использован, если система не существует или недоступна, например в начале проектирования. В этих случаях, так как правильность модели не может быть установлена эмпирически, наша вера в модель должна основываться на нашем доверии к правильности ее исходных концепций и конструкции.

Часто в работе по оценке используются несколько методов оценки. Различие характеристик разных методов (которые будут обсуждены в гл. 2, 3 и 4) приводит к тому, что каждый из них может оказаться удобнее, чем другие методы, в зависимости от различных типов проблем, часто встречающихся в одном и том же исследовании. К тому же использование в работе различных подходов, методов и приемов позволяет оценщику достичь лучшего понимания системы.

Общей проблемой для всех методов оценки вычислительных систем является *описание рабочей нагрузки*. Рабочая нагрузка вычислительной системы уже была определена в разд. 1.2 как множество входных сигналов, создаваемых ее коллективом пользователей. Как будет показано в гл. 5, рабочую нагрузку необходимо описывать как при моделировании, так и при проведении измерений. При измерениях следует контролировать рабочую нагрузку, чтобы обеспечить воспроизводимость эксперимента. Эта проблема, как показано в гл. 5, концептуально похожа на проблему конструирования моделей рабочей нагрузки при моделировании. Вследствие ее решающей роли в оценке производительности, описание рабочей нагрузки является одной из главных тем этой книги.

Задачи

1.1. В предположении, что у ПРОП-системы, описанной в примере 1.1, имеется только один пользователь, который последовательно вводит шесть команд $c_1, \ldots, c_6$ при постоянном времени обдумывания 5 секунд, определите и вычислите $\bar{t}$ и $\bar{T}$ при помощи

a) построения временной диаграммы, как на рис. 1.6;

b) выписывания аналитических формул и использования их. Влияет ли продолжительность кванта времени на производительность системы? Необходим ли квант времени в системе с одним пользователем? А в диалоговой однопрограммной системе с несколькими пользователями?

1.2. Определите единый индекс производительности, объединяющий оба индекса, введенные для ПРОП-системы в примере 1.1. Определите в соответствии с новым индексом оптимальную величину кванта времени q для рабочей нагрузки, описанной на рис. 1.6.

1.3. Определите по крайней мере два индекса, которые вы выбрали бы для оценки производительности программы, которую вы недавно написали. Предполагая, что вам придется делать эту программу заново, напишите ее спецификации производительности в терминах выбранных индексов. Припишите индексам относительные веса и обоснуйте выбор этих весов.

1.4. Определите коэффициент использования центрального процессора ρ_1 для диалоговой ПРОП-системы, описанной в примере 1.1, как отношение суммарного времени обработки команд пользователя к общему времени, требуемому для завершения обработки всей рабочей нагрузки. Напишите аналитическую формулу для вычисления ρ_1 при тех же предположениях, которые были сделаны при выводе уравнений (1.1) и (1.2). Выведите соотношение между ρ_1 и $\bar{T}$ и постройте график ρ_1 от q.

1.5. Перечислите различные причины, по которым центральный процессор диалоговой ПРОП-системы, описанной в примере 1.1, может оказаться незанятым. Определите коэффициент использования центрального процессора ρ_1 как отношение времени, когда центральный процессор работает (включая накладные расходы), к общему времени обработки n команд, оставляющих рабочую нагрузку системы. Выведите из определений ρ_1 и $\bar{T}$ (среднее число команд, обработанных в единицу времени) соотношение между этими двумя индексами. Покажите, используя полученное вами соотношение, как определенные изменения в системе могут увеличивать ρ_1, уменьшая $\bar{T}$.

1.6. Иногда используется индекс продуктивности системы, определяемый как средняя частота выполнения команд, измеряемая в командах в секунду. Этот индекс (который будет обозначаться $\bar{v}_{цп}$) не следует путать с максимальной частотой выполнения команд центральным процессором $v_{цп\,max}$. Каково соотношение между $\bar{v}$ и $v_{цп\,max}$ (если оно есть) и коэффициентом использования центрального процессора? Ожидали бы вы непременного повышения $v_{цп}$ при замене центрального процессора на другой с тем же набором команд, но с бо́льшим значением $v_{цп\,max}$?

1.7. Простая однопрограммная система пакетной обработки состоит из центрального процессора, основной памяти, устройства чтения перфокарт, устройства печати и четырех лентопротяжных устройств. Два лентопротяжных устройства используют-

Рис. З.1.1. Система пакетной обработки, рассмотренная в задаче 1.7.

ся для ввода и два — для вывода. Пока пакет читается и накапливается на одной ленте, центральный процессор может переносить задания поодиночке с другой входной ленты (содержащей предыдущий пакет) в память. Аналогично, в то время как результаты некоторого задания печатаются с одной из выходных лент, центральный процессор может записывать результаты следующего задания на другую выходную ленту. Блок-схема этой системы и скорости передачи показаны на рис. З.1.1. Предположим, что все задания вводятся в систему на машинном языке, что каждые 30 минут в устройство чтения перфокарт поступает очередной пакет, что каждый пакет состоит из 60 заданий: 55 заданий типа 1 и 5 типа 2 (см. табл. З.1.1), и что центральный процес-

сор полностью свободен к моменту окончания загрузки пакета на входную ленту. Подсчитайте среднюю пропускную способность (т. е. среднее число заданий, обработанных в единицу времени) за период много больше 30 минут. Подсчитайте также коэффициент использования центрального процессора и максимальное время прохождения, определенное как наибольший возможный интервал между моментом вручения колоды карт оператору и моментом получения напечатанных результатов. Предполагая, что все задания типа 1 предшествуют в пакете всем заданиям типа 2, подсчитайте среднее время пребывания задания в системе, определенное как время между моментом окончания чтения пакета этого задания в устройстве чтения перфокарт и моментом появления выдачи от задания. Определим теперь *среднее задание* как задание, характеристики которого являются взвешенными средними от характеристик в табл. 3.1.1. Рассмотрите пакет, составленный из 60 средних заданий. Совпадает ли среднее время пребывания этого пакета в системе с временем пребывания исходного пакета? Почему?

Таблица 3.1.1

Характеристики рабочей нагрузки для задачи 1.7

Параметр задания	Тип 1 (55 заданий)	Тип 2 (5 заданий)
Прочитано перфокарт	200	500
Размер (слов)	12000	30000
Время выполнения (с)	15	90
Слов выведено	4500	1500
Строк напечатано	300	100

1.8. В пакетной ПРОП-системе все задания идентичны (см. рис. 1.2). Каждое задание состоит из одного шага задания. Загрузка задания в память занимает 0.5 с. Выполнение задания занимает 10 интервалов центрального процессора, каждый длительностью 10 мс, разделенных обращениями к барабану, каждое по 30 мс. В конце последнего интервала центрального процессора выходные данные задания переносятся на диск за 0.2 с. Предполагается, что поток заданий, ожидающих на диске своей очереди на загрузку в память, никогда не оканчивается. Подсчитайте среднюю пропускную способность системы за продолжительный период времени (скажем, за 1 день) для степеней мультипрограммирования от 1 (однопрограммная система) до 5. Вычислите также (при тех же самых условиях) время среднего пребывания задания в памяти, т. е. среднее время между моментом, когда задание полностью загружено в память, и моментом, когда все выходные данные перенесены на диск. Какова цена увеличения пропускной способности, полученной за счет увеличения степени мультипрограммирования?

2
Методы измерения

2.1. РАБОТЫ ПО ИЗМЕРЕНИЮ

Одно из решений, которые приходится принимать на этапе планирования работ по оценке, относится к выбору метода (или методов) для использования. Если выбран эмпирический метод, главные цели этапа планирования заключаются в

a) решении, что измерять,
b) выборе измерителя (или измерителей),
c) планировании экспериментов и оценке их стоимости.

На этапе выполнения плана проводятся эксперименты, получаются и накапливаются данные. Наконец, на этапе интерпретации данные анализируются оценщиками. В разд. с 2.2 по 2.5 мы опишем измерители и обсудим их выбор и критерии планирования. Проблемы, которые возникают при планировании измерительных экспериментов, частично охватываются разд. 2.6, частично гл. 5, в которой обсуждаются проблемы выбора рабочей нагрузки. Наконец, в разд. 2.7 мы проиллюстрируем некоторые методы сжатия данных и методы анализа. В оставшейся части этого раздела кратко исследована проблема определения, что может быть измерено и что следует измерять.

Что измеряют на самом деле измерители вычислительных систем? Их целью является измерение индексов производительности и параметров установки. Любой измерительный эксперимент можно рассматривать как основанный на мысленном представлении системы через вход — выход. Эта мысленная модель относится к типу внешнее воздействие — ответ. Ясное сознание этого поможет нам планировать эксперименты корректно.

Предположим, что мы хотим измерить в системе пакетной обработки индекс производительности, например среднюю пропускную способность $\overline{T}$, и параметр установки, например среднее время поиска на диске $\overline{t}_{\text{поиск}}$. Средняя пропускная способность есть индекс производительности, обычно определяемый как число заданий, обработанных в единицу времени на протяжении заданного интервала. Время поиска для диска с подвижными головками есть время, затраченное на подвод головок к требуемому цилиндру. Среднее $\overline{t}_{\text{поиск}}$ придется вычислять по множеству попыток, в нашем случае произведенных в течение интервала, за который измерено $\overline{T}$.

Как $\overline{T}$, так и $\overline{t}_{\text{поиск}}$ зависят от рабочей нагрузки, обрабатываемой системой во время эксперимента. Таким образом, ситуация соответствует рис. 2.1 (a). Концептуальная модель на этом рисунке концентрирует наше внимание на внешнем воздействии, на которое система «отвечает» значениями $\overline{T}$ и $\overline{t}_{\text{поиск}}$, которые мы и собираемся измерять. Для того чтобы эксперимент был корректным и значимым, это воздействие следует контролировать или по крайней мере наблюдать.

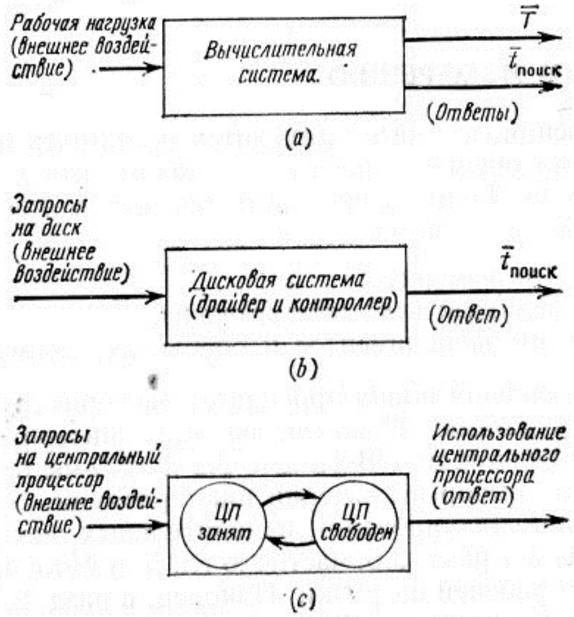


Рис. 2.1. Концептуальные модели, лежащие в основе некоторых измерительных экспериментов.

Другая концептуальная модель для измерения $\overline{t}_{\text{поиск}}$ представлена на рис. 2.1 (b). В основе измерения коэффициента использования центрального процессора лежит модель с двумя состояниями, изображенная на рис. 2.1 (c).

Из-за дискретной интерпретации, которую мы обычно даем поведению вычислительной системы, измерители по большей части измеряют индексы и параметры посредством обнаружения некоторых *событий*. Событие есть любое изменение в системе или в рабочей нагрузке, интересующее оценщика. В контексте описания установки в терминах состояний события соответствуют переходам между состояниями.

В вычислительных системах изменения могут сильно различаться по степени детальности и масштабу времени. Например, изменение уровня напряжения с высокого на низкий (или наоборот), появление импульса в линии, выполнение предписанной команды, появление определенной комбинации битов в регистре, окончание поиска на диске, инициация выполнения задания, поступление команды с терминала — все они могут быть определены как события в одной и той же системе.

Во многих случаях измерение индексов производительности и параметров установки в действительности является вычислением, которое выполняется в момент обнаружения события или позже, на основе собранных данных. Таким образом, многие измерения являются *косвенными*, так как их результаты получены после процесса сжатия данных. Например, средняя пропускная способность $\overline{T}$ установки с пакетной обработкой может быть получена путем подсчета числа заданий, покидающих систему в течение заданного интервала времени. В этом случае нужен измеритель, способный обнаруживать окончание каждого задания. Подсчет этих событий может быть выполнен в момент обнаружения или, если возникновение каждого события регистрируется в некотором накопителе, позже. Заметим, что те индексы и параметры, которые доступны для этого измерителя в момент обнаружения событий, могут быть измерены непосредственно.

Измеритель, которому предстоит обнаруживать событие, должен быть способен отделять соответствующее изменение от всех остальных как на разных уровнях детальности, так и на том же самом уровне. *Избирательность* в реальном времени позволяет измерителю фильтровать события, отбрасывая ненужную информацию при сборе данных, вместо того чтобы записывать все изменения и затем отбирать относящиеся к делу.

Что касается типа изменений, определенных как события, следует различать случаи измерения *внутренних* событий и измерения *внешних* событий. Эта классификация подразумевает, что границы системы определены точно. Если мы рассмотрим всю установку в качестве объекта измерений, внешним событием может быть только некоторое наше действие или наступление заранее определенного момента времени. Обнаружение моментов внешних событий часто используется для *выборки* значений индексов и параметров установки.

Как только определены интересующие нас индексы производительности и параметры установки, обычно легко установить, какие события должны обнаруживаться нашими измерителями и на этом основании какие типы измерителей следует использовать. Гораздо более трудной проблемой оказывается принятие решения о выборе измеряемых индексов параметров.

Некоторые общие соображения о выборе индексов производительности были высказаны в разд. 1.4. Интересующие нас параметры зависят от типа установки и системы, а также от задачи оценки, с которой мы имеем дело. Несколько примеров задач, к которым могут быть применены методы измерения, обсуждается в гл. 6, 7 и 8. Вообще, оценщики заинтересованы в измерении тех параметров, которые могли бы помочь им лучше понять поведение системы. Например, значения некоторых параметров могут обнаруживать причины неправильной работы системы. В примере, рассмотренном выше, значение $\overline{t}_{\text{поиск}}$, большее ожидаемого, наводило бы на мысль, что одной из причин того, что значение $\overline{T}$ меньше ожидаемого, возможно является неэффективная организация информации на диске.

2.2. ВВЕДЕНИЕ В ИЗМЕРИТЕЛИ

Каждый измеритель в принципе состоит из трех частей, выполняющих различные функции (рис. 2.2): *датчика*, задачей которого является восприятие размера измеряемой величины; *преобразователя*, способного осуществлять желаемое преобразование информации, которую он получает от датчика; *индикатора*, который позволяет экспериментатору прочесть результат измерения. Например, измерительная линейка

Рис. 2.2. Три принципиальных элемента измерителя.

воспринимает (или позволяет тому, кто ею пользуется, воспринять) деление, соответствующее концу объекта, длина которого измеряется, и преобразует в число, показывающее в некотором общепринятом масштабе длину объекта. Аналогично, термометр воспринимает температуру окружающей среды и преобразует ее в удлинение или сокращение столбика ртути, высота которого в градуированном капилляре показывает в общепринятом масштабе значение температуры.

Измерители в вычислительных системах часто обнаруживают события с гораздо большей частотой, чем может прочесть человек. Поэтому индикатор обычно накапливает результаты измерений в некотором накопителе вместо того, чтобы показывать их экспериментатору в реальном времени. Это не означает, что результаты измерений вообще не могут быть показаны в реальном времени. Такой показ возможен, если наблюдаемые события относительно редки или если собранная информация периодически выдается на выход (после соответствующего сжатия) в ходе эксперимента. Заметим, что преобразователь может ничего не делать, тогда события записываются без преобразования. С другой стороны, он может предварительно обрабатывать результат, например прибавлять 1 к счетчику событий при каждом обнаружении события. Иногда преобразователь выполняет также и другие действия; например, он считывает показания часов или содержимое некоторого регистра, или содержимое одной или нескольких ячеек памяти, или реализует любую комбинацию этих возможностей.

Наиболее важные характеристики измерителей, на которых основываются их выбор и спецификации при разработке, перечислены (но не по порядку важности) в табл. 2.1. Обсудим эти характеристики вкратце.

Если измеритель использует ресурсы системы, его работа может мешать работе системы и оказывать заметное влияние на измеряемые величины. Таким образом, *помеха* может вызывать как снижение производительности, так и неточность в результатах измерений. *Снижение* производительности есть любое уменьшение ее по сравнению с уровнем, который мог бы быть достигнут при той же самой пользовательской рабочей нагрузке. Неточности, которые могут быть вызваны помехой, имеют определенно более серьезное значение, по крайней мере для экспериментатора.

Таблица 2.1

Основные характеристики измерителей

Уровень помех
Точность
Разрешающая способность
Область применения
Возможности предварительной обработки
Совместимость
Стоимость
Простота установки
Простота пользования

Точность измерителя может быть оценена величиной ошибки, вносимой им в данные. Существует несколько источников ошибок, которые могут обусловить отличие измеренного значения некоторой величины от истинного ее значения. Взаимодействие измерителя с системой — один из таких источников. Остальные ошибки могут быть обусловлены его неправильным подключением или использованием, неточной работой или ограничениями его датчика (детектора событий) или преобразователя. Одним из таких ограничений является число разрядов, используемых для представления собранной информации. Это ограничение на *точность* измерителя. Другое ограничение относится к его *разрешающей способности*, т. е. к максимальной частоте появления событий, при которой события еще могут быть обнаружены и правильно записаны.

Ошибки измерения могут быть также следствием плохой работы или ограниченной точности других измерителей (например, часов), участвующих в эксперименте. Дополнительные ошибки могут возникнуть на этапах сжатия информации, ее выдачи и интерпретации. Анализ ошибок эксперимента — сложный и крайне важный вид деятельности. В этой главе мы обсудим некоторые его аспекты в связи с некоторыми особыми измерителями (см. примеры 2.1—2.4 и 2.6), а также в связи с общими проблемами планирования эксперимента и интерпретации результатов (разд. 2.6 и 2.7).

Область применения измерителя определяется теми классами событий, которые он может обнаружить. Эту характеристику можно интерпретировать различными способами, например, как описывающую гибкость измерителя или диапазон его возможных применений, или возможность переноса с системы на систему. Здесь следует отметить, что в систему могут быть встроены измерители, область применения которых ограничена одним или несколькими типами событий, и экспериментатор не может изменить эти типы. Они будут называться *фиксированными измерителями*. Так как гибкость и общность обычно обходятся дорого, большинство измерителей, используемых при не-

прерывном наблюдении (см. разд. 1.3), относится к фиксированным измерителям.

Преобразователь, входящий в состав измерителя, может обрабатывать собранные данные. Следовательно, первое сжатие информации, которое будет называться *предварительным сжатием*, может быть выполнено до запоминания данных. Это верно для *счетчиков*, подсчитывающих события. Если нет предварительного сжатия, то мы имеем *следящий измеритель*. Это измеритель, записывающий последовательность событий в хронологическом порядке. Такая последовательность называется трассировочной записью. Следящий измеритель способен отфильтровывать интересные события, игнорируя все остальные изменения. Однако он не обрабатывает собранные данные.

Предварительное сжатие экономит память, время процессора ввода-вывода и часть процессорного времени, необходимого на последующее сжатие в автономном режиме. С другой стороны, оно делает измеритель более дорогим за счет дополнительной аппаратуры или программного обеспечения. Оно также увеличивает помехи или снижает разрешающую способность, или и то и другое. Предварительное сжатие разрушает часть информации, которая могла бы оказаться полезной позже, особенно если эксперимент не был заранее тщательно спланирован. Таким образом, выбор должной степени предварительного сжатия информации сводится к компромиссу, что будет исследовано в разд. 2.4.1 для частного случая.

Совместимость относится к различным аспектам системы, которую предстоит измерять. Измеритель, предназначенный для обнаружения импульсов напряжения, не должен ими повреждаться и должен соответствующим образом реагировать на их фронты и продолжительность. Измеритель, состоящий из набора программ, должен корректно взаимодействовать с операционной системой. Все измерители должны быть также совместимы с защитными механизмами измеряемой системы. Эти вопросы еще будут подробнее рассматриваться в данной главе.

Стоимость измерителя включает затраты на его приобретение, аренду (или разработку и изготовление); стоимость установки и технического обслуживания, обучения пользователей, эксплуатационные расходы. Важность последних двух характеристик из перечисленных в табл. 2.1 очевидна и в дополнительных комментариях не нуждается. Однако было бы полезно подчеркнуть, что наличие ясной, недвусмысленной, исчерпывающей документации для измерителей также существенно.

По своей природе измерители традиционно делятся на классы *аппаратных измерителей*, *программных измерителей* и *микропрограммных измерителей*. Это различие не касается природы измеряемых величин; в принципе непрерывное наблюдение за программой можно выполнять аппаратным измерителем, а производительность аппаратного модуля может измеряться программным измерителем. Действительно, аппаратный измеритель состоит из оборудования, добавляемого к системе с целью измерения. Программные измерители состоят

из команд, которые должна выполнять измеряемая система. Микропрограммные измерители состоят из микрокоманд, которые также должна выполнять измеряемая система, причем предполагается, что она содержит, как минимум, один микропрограммируемый процессор. Расплывчатость границ между аппаратурой, программным и микропрограммным обеспечением делает это различие довольно нечетким, тем более, что в последнее время появляются *гибридные измерители*. Эти измерители (см. разд. 2.4.1) требуют добавления к измеряемой системе как дополнительного оборудования, так и дополнительного программного обеспечения. Увеличивается использование в качестве измерителей вычислительных машин общего назначения. Мы будем относить их к аппаратным измерителям, хотя они в действительности являются аппаратно-программными комплексами.

Другое важное различие, которое следует упомянуть, связано с тем, являются ли обнаруживаемые события внутренними или внешними. В первом случае измеритель называется *внутренне управляемым*. В последнем случае мы имеем *внешне управляемый* измеритель, который обычно относится к типу *выборочных* измерителей.

2.3. АППАРАТНЫЕ ИЗМЕРИТЕЛИ

2.3.1. Введение

Аппаратные измерители, особенно внутренне управляемые, обнаруживают события, происходящие на относительно микроскопическом уровне, например наличие импульса электрического напряжения или фронта импульса на определенной шине или появление определенной последовательности битов в регистре. Это обусловливает высокую разрешающую способность таких измерителей, которые могут обнаруживать высокочастотные микроскопические события. Их область применения потенциально широка, так как большая часть интересующих исследователя точек системы обычно доступна снаружи, несмотря на то что тенденция к увеличению степени интеграции элементов систем делает все большее количество сигналов недоступными. Итак, интересующие исследователя точки доступны, особенно если требования измерения системы были приняты во внимание еще на стадии разработки системы. Конечно, реальная область применения зависит от конкретного типа измерителя. Вообще обнаружение более макроскопических событий и соотнесение микроскопических событий с их причинами (например, с порождающими их заданиями) — более трудные проблемы для большинства аппаратных измерителей.

То, что эти измерители состоят из дополнительного оборудования, делает помехи от них малыми или вовсе сводит к нулю. Ненулевые помехи возникают тогда, когда собранные данные периодически запоминаются в измеряемой системе для немедленного или отложенного сжатия или когда измерителю разрешен доступ в основную память измеряемой машины. Помехи могут увеличиться, если существенная

часть ресурсов измеряемой системы (например, один процессор, часть основной и внешней памяти) используется для целей измерения.

Точность и возможности предварительного сжатия данных меняются от измерителя к измерителю; они будут обсуждены для конкретных типов измерителей в следующих разделах. Спектр цен на аппаратные измерители настолько же широк, насколько разнообразны их типы и особенности. Их установка требует обычно обширной экспертизы и детального знания измеряемой системы, а их пользователи должны быть хорошо обучены.

Почти все аппаратные измерители для сжатия собранной информации должны обладать вычислительной мощностью компьютера. Иногда эта мощность обеспечивается за счет самой измеряемой системы или во время эксперимента (в этом случае часто помехи достаточно велики), или по окончании эксперимента. В других случаях сжатие информации выполняется внешним компьютером, который может быть непосредственно соединен с измерителем и может производить сжатие данных в реальном времени. Если компьютер используется только для сжатия данных и не управляет экспериментом, измеритель называется *измерителем с фиксированной программой*. *Измеритель с запоминаемой программой* состоит из компьютера, должным образом соединенного с измеряемой системой, программное обеспечение которого управляет обнаружением событий, сбором информации, ее обработкой и накоплением.

Эти два типа аппаратных измерителей описаны в разд. 2.3.3 и 2.3.4 соответственно. В следующем разделе кратко обсуждаются фиксированные аппаратные измерители.

2.3.2. Фиксированные аппаратные измерители

Фиксированный измеритель есть инструмент, показания которого могут быть прочитаны пользователями, но сам он ими не управляется. Одним из аспектов этой недостаточной управляемости оказывается то, что пользователи не могут выбирать ни типы обнаруживаемых измерителем событий, ни возможные действия, которые он выполняет при обнаружении события. Следовательно, область применения фиксированного измерителя крайне ограничена.

Фиксированные аппаратные измерители обычно включаются в систему при ее создании. Добавление такого измерителя к готовой работающей системе, вообще говоря, не просто. Существуют измерители, *читаемые оператором* и *читаемые программно*. К первой категории относятся часы в машинном зале и те регистры или счетчики, встроенные в систему для отладки и измерения, содержимое которых выдается на консоль оператора. Ко второй категории относятся программно читаемые системные часы и те аппаратные счетчики (если они есть), которые непрерывно подсчитывают события определенного типа и могут быть прочитаны специальными командами. Эти категории раз-

делются не абсолютно; например, обычно оператор может прочесть показания системных часов.

Пример 2.1. Требуется измерить общее время $t_{общ}$, необходимое системе для обработки пакета из N заданий. Мы можем определить среднюю пропускную способность системы при обработке этих N заданий как

$$\overline{T} = \frac{N}{t_{общ}}. \qquad (2.1)$$

Таким образом, измерение $t_{общ}$ позволит нам косвенно и измерить $\overline{T}$, применяя (2.1). Нам достаточно лишь точно определить моменты начала и конца выполнения пакета заданий, и мы можем считывать показания часов в машинном зале, системных часов или своих часов в эти моменты. Разность между этими моментами времени дает значение $t_{общ}$. Это измерение $t_{общ}$ также является косвенным. Прямое измерение требует секундомера или эквивалентного ему измерителя, например счетчика, который в начале эксперимента устанавливается в нулевое значение и показания которого будут считаны в конце.

Результат даже такого простого измерения, как это, может быть подвержен некоторым ошибкам. Среди них мы упомянем ошибки, обусловленные ограниченной точностью часов, неточным считыванием их показаний, ошибкой вычисления разности двух записанных времен, неточным обнаружением моментов появления событий (например, момент второго считывания часов может не совпасть с моментом окончания обработки пакета). Полезно заметить в отношении систематических ошибок, воздействующих на оба отсчета в одну сторону, что на значение $t_{общ}$ влияет лишь их разность.

Другой, потенциально более точный метод проведения того же самого измерения использует возможность программного чтения часов системы. Операционной системе может быть дана команда считывать показания этих часов в моменты, определенные как момент инициации и момент завершения выполнения данного пакета. В другом варианте каждое задание может считывать показания часов в начале и в конце выполнения и выдавать эти два отсчета. Все перечисленные выше источники ошибок могут влиять в принципе и на эти измерения. Однако в этом случае ответственность за неточное считывание показаний часов и за неодновременность ложится на процедуру доступа к часам.

Во многих системах считывание показаний часов требует времени, особенно если это делается по заданию пользователя. Другим источником ошибок являются помехи, обусловленные считыванием показаний часов и выдачей результатов. Часто удается ввести коррекцию, полностью или частично компенсирующую эти ошибки. Читатель может сам придумать модификации, которые следует внести в операционную систему или в задание пользователя для чтения часов и выдачи результата; во всяком случае, эти модификации описаны в разд. 2.4.2. Заметим, что практически все операционные системы выполняют эти измерения программным способом для целей учета. ■

Многие современные системы оборудованы программно доступными измерителями, которые могут генерировать внешние события. Например, система IBM/370 кроме 64-битовых часов реального времени, отсчитывающих каждую микросекунду, имеет таймер центрального процессора и интервальный таймер. Таймер центрального процессора представляет собой 64-битовый регистр, загружаемый из программы и уменьшающий свое содержимое каждую микросекунду. Когда его содержимое становится отрицательным, генерируется внешнее прерывание центрального процессора. Интервальный таймер есть фиксированная ячейка основной памяти, содержимое которой загружается программой и уменьшается на единицу каждые 3.3 миллисекунды. Ана-

логично, когда его содержимое становится отрицательным, генерируется внешнее прерывание. Эти два таймера могут использоваться в качестве часов, так же как и часы реального времени. В оборудование системы 370 и других систем входит компаратор часов. В системе 370 это 64-битовый регистр, в который программа может загружать последовательность битов, интерпретируемую как значение реального времени. Когда содержимое часов реального времени становится равным этому значению, генерируется прерывание. В измерительных экспериментах этот измеритель может быть использован для пуска или остановки сеанса измерений в заданное время дня. Как компаратор часов, так и таймеры (при их использовании для генерации отсроченных прерываний) являются датчиками внешних событий. Однако таймеры при использовании их в качестве часов можно рассматривать как фиксированные измерители.

2.3.3. Измерители с фиксированной программой

Блок-схема простого измерителя с фиксированной программой приведена на рис. 2.3. *Зонды* — это детекторы уровня напряжения с высоким входным сопротивлением, связанные с соответствующими точками схемы. Высокое входное сопротивление нужно, чтобы избежать

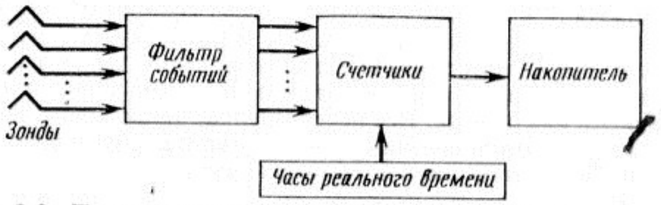


Рис. 2.3. Простой измеритель с фиксированной программой.

нарушения работы цепей, к которым они присоединены. Сигналы (уровни напряжения и импульсы), собранные зондами, обрабатываются *фильтром событий*, обычно набором логических элементов, соединения которых могут быть реализованы на коммутационной панели самим пользователем. Эти элементы могут включать в себя параллельные и последовательные цепи и иногда бывают сменными. Заметим, что такой фильтр можно реализовать на постоянной памяти (предназначаемой только для считывания). Эта память может быть загружаемой или сменной, если требуется гибкий измеритель. Она должна быть быстрой, чтобы серьезно не ограничивать разрешающую способность измерителя.

Выходы фильтра, так же как и выход часов реального времени, присоединены к набору *счетчиков*. Каждый счетчик может использоваться в двух режимах: в *режиме счета* (подсчитываются импульсы, приходящие от логических элементов) и в *режиме таймера* (разрешение на счет, поступающее от логических элементов, открывает путь импульсам от часов на вход счетчика) (см. рис. 2.4).

Счетчики в режиме таймеров могут быть использованы для измерения длительностей импульсов, значительно превышающих период часов. Этот механизм проиллюстрирован на рис. 2.5, который поясняет, как можно измерить коэффициент использования компонента вычислительной системы посредством измерителя, подобного изображенно-

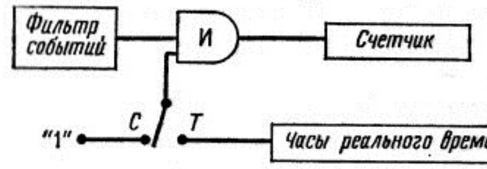


Рис. 2.4. Два режима работы счетчика: *С* — режим счета, *Т* — режим таймера.

му на рис. 2.3. Если в конце сеанса измерения содержимое двух счетчиков А и В (которые были очищены в начале сеанса) равно соответственно C_A и C_B, оценка для коэффициента использования измеряемого блока будет C_A/C_B.

Содержимое этих счетчиков и часов реального времени периодически разгружается в *накопитель*. Накопитель в большинстве случаев

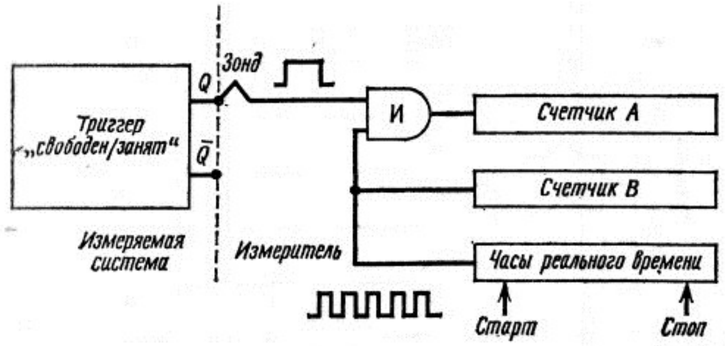


Рис. 2.5. Измерение коэффициента использования с помощью аппаратного измерителя.

реализуется на магнитной ленте. Эта лента будет затем обрабатываться на измеряемом или на другом компьютере с целью сжатия данных и генерации отчета. Некоторые ранние измерители имели визуальный индикатор вместо накопителя. Индикатор полезен в дополнение к накопителю, чтобы контролировать эксперимент. Частота разгрузок на ленту обычно может быть выбрана экспериментатором в определенных пределах. Импульсы разгрузки генерируются часами, имеющими несколько выходов с различными частотами. В некоторых измерителях эти часы могут быть также синхронизированы с внешним генератором импульсов, например с часами измеряемой системы.

Хотя измерители с фиксированной программой использовались и много раньше отдельными изготовителями [например, Программный монитор, описанный Эпплом (1965), и TS/SPAR, описанный Шульманом (1967), оба разработаны в IBM], на рынке они появились в 1968 году. Их эволюция с тех пор протекала быстро, все более приб-

лижая их к измерителям с запоминаемой программой. Главными добавлениями были: параллельный вход и память с произвольным доступом. *Параллельный вход* состоит из набора зондов, которые могут считывать содержимое любого доступного регистра измеряемой системы. Это содержимое может непрерывно сравниваться со значениями, загруженными пользователем в регистры измерителя. Результаты этих сравнений можно переслать затем на счетчики, работающие в режиме счета или таймера.

Память произвольного доступа (ППД) может заменить счетчики, при условии что можно допустить неминуемое снижение скорости счета. При наличии параллельного входа можно заставить ППД работать в *режиме распределения*, при котором гистограмма последовательности событий, принадлежащих к нескольким классам, строится инкрементно. Это обеспечивается посредством адресации памяти с использованием кодированных должным образом меток событий или с помощью

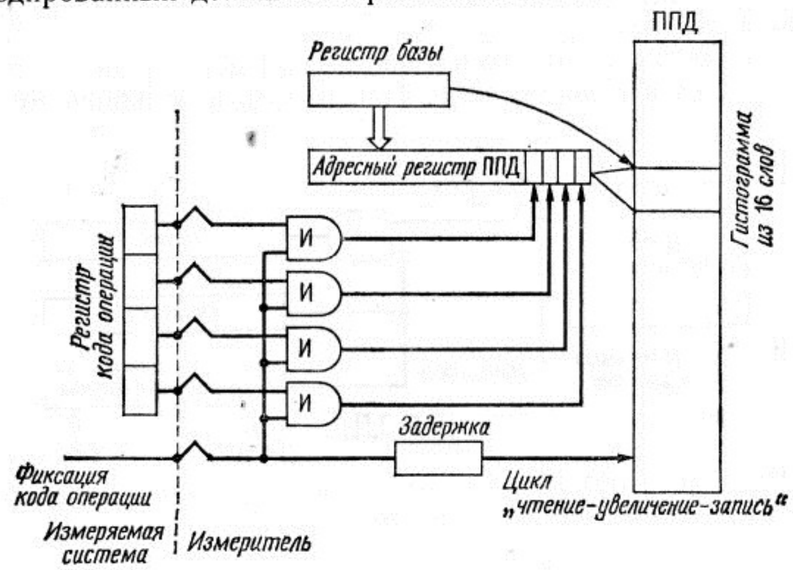


Рис. 2.6. Использование параллельного входа и памяти произвольного доступа в режиме распределения.

иных подходящих данных, собранных в момент события. Простой пример приведен на рис. 2.6, который показывает экспериментальную структуру для получения гистограммы типов команд, выполняемых в течение сеанса измерения. Событие, которое надо обнаружить, происходит тогда, когда приходит сигнал по линии «код операции». При каждом обращении к ППД выбранное слово увеличивается на 1 и запоминается вновь. В каждый момент времени вектор из 16 последовательных слов в памяти, начальный адрес которых образуется содержимым базового регистра и далее четырьмя нулями в младших разрядах, содержит гистограмму кодов операций, выполненных до данного момента. Следует заметить, что цикл «чтение — увеличение — запись» ППД должен быть короче, чем минимальный интервал между

двумя последовательными выборками команд; иначе может быть потеряна часть информации.

Другим способом использования параллельного входа и ППД является режим *трассировки*, при котором комбинации битов, поступающие в измеритель по параллельным входам, запоминаются в последовательности ячеек ППД. Периодически трассировочная запись, накопленная в ППД, должна разгружаться на вторичный накопитель, если только сеанс измерения не слишком короток. Тогда в качестве буфера для накопленных данных может быть использован ППД.

В режиме трассировки существует также очевидная нижняя граница для скорости ППД. Эта граница зависит от средней и максимальной частот появления обнаруживаемых событий. Трассировочные записи, вообще говоря, требуют большого объема памяти и, следовательно, возможность их накопления ограничена короткими периодами времени.

Когда надо обнаруживать только заранее указанные последовательности событий, нужда в трассировке может отпасть при использовании *детекторов последовательностей*. Эти детекторы являются последовательными цепями, которые опознают заданные пользователем последовательности. Например, предположим, что мы хотим измерять количество случаев, когда выполняются две или более команд передачи управления подряд в течение определенного интервала времени. Предположим для простоты, что все команды передачи управления имеют один и тот же код операции. Таким образом, проблема сводится к распознаванию в потоке команд всех случаев появления подряд двух или более кодов команд передачи управления. Для решения задачи можно использовать последовательную цепь с тремя состояниями. Появление кода операции передачи управления будет вызывать переход из состояния S_0 в состояние S_1. Следующий код операции (если это код операции передачи управления) вызовет переход в состояние S_2 и появление выходного импульса, который увеличит счетчик на единицу; если нет, он изменит состояние опять на S_0. В состоянии S_2 код операции передачи управления оставит детектор последовательности в состоянии S_2 без выдачи дополнительного импульса, тогда как код, отличный от кода передачи управления, изменит состояние на S_0.

В табл. 2.2 дана сводка основных характеристик некоторых коммерческих измерителей с фиксированной программой.

Возможно, самым типичным применением измерителей с фиксированной программой является измерение коэффициентов перекрытия и использования. Обобщение структуры, показанной на рис. 2.5, описано в следующем примере.

Пример 2.2. Необходимо провести измерения в ПРОП-системе (см. рис. 1.2). Оценщиков интересует среди прочих величин активность центрального процессора и процессора барабана (для краткости назовем его *канал*). К основным вопросам относятся распределение времени центрального процессора и канала, а также временные соотношения их активности. Для целей этого эксперимента определим три состояния

Таблица 2.2

Технические характеристики некоторых коммерческих аппаратных измерителей с фиксированной программой

Изготовитель	Allied Computer Technology	Computer Synectics	Tesdata Systems	Tesdata Systems
Модель	CPM-II	SUM	1010	1155
Зонды				
количество	20, 40	20	20	32, 96
миним. длит. импульса (нс)	50	50	20	30
макс. частота (МГц)	5	10	20	15
Логика (коммутационная доска)	450, 600 контактов	300 контактов	200 контактов	480, 960 контактов
Счетчики				
количество	16, 32	16, 32	8, 16	8, 32
длина	10 десятичных цифр	9 десятичных цифр	32 бит	32 бит
макс. частота (МГц)	5	2	20	3.5
Компараторы	1 (не обязат.)	1, 5 (не обязат.)	—	—
Детекторы последовательностей	Нет	Нет	Нет	Да
Режим распределения	Нет	Нет	Нет	Да
Память произвольного доступа	Нет	Нет	Нет	Да
длина слова (бит)	—	—	—	16
размер (байт)	—	—	—	4 *K*, 8 *K*
цикл (мкс)	—	—	—	1
Накопитель	Одна 9-дорожечная лента	Одна 7- или 9-дорожечная лента	1 кассета (не обязат.)	Одна 7- или 9-дорожечная лента
Периферийные устройства	Видеодисплей	Видеодисплей	Дисплей Поколонная печать (не обязат.)	—
Программное обеспечение	MSR (генератор суммарных отчетов)	SUMDAP (анализ данных)	Пакетный анализатор	Пакетный анализатор

центрального процессора и два состояния канала. В любой данный момент центральный процессор либо *занят*, либо *свободен*, а если занят, то может находиться либо в состоянии *супервизора*, либо в состоянии *задачи*. Центральный процессор находится в состоянии задачи, когда он обрабатывает задание пользователя, а в состоянии супервизора, когда работает операционная система. Аналогично, канал может быть в каждый момент либо занят, либо свободен. Состояния пары «центральный процессор — канал» могут быть определены как комбинации состояний отдельных устройств. Если

считать, что центральный процессор имеет три состояния, будет всего шесть возможных комбинаций. Если предположить только два состояния для центрального процессора (занят и свободен), тогда возможных комбинаций окажется четыре.

Предположим, что оценщиков прежде всего интересуют эти четыре состояния, а также разбиение времени, в течение которого процессор находится в состоянии занят, на две части, соответствующие состояниям: задача и супервизор. Длительность состояния «центральный процессор занят/канал занят» является мерой *перекрытия центральный процессор/канал*. Состояния «процессор занят/канал свободен» и «процессор свободен/канал занят» обозначаются соответственно терминами *только ЦП* и *только канал*. Наконец, состояние «процессор свободен/канал свободен» называется *свободным состоянием*.

Если триггеры «свободен/занят» обоих устройств и триггер «задача/супервизор» центрального процессора доступны, то достаточно трех зондов, соединенных с выходами этих триггеров, чтобы измерять длительности всех состояний на протяжении

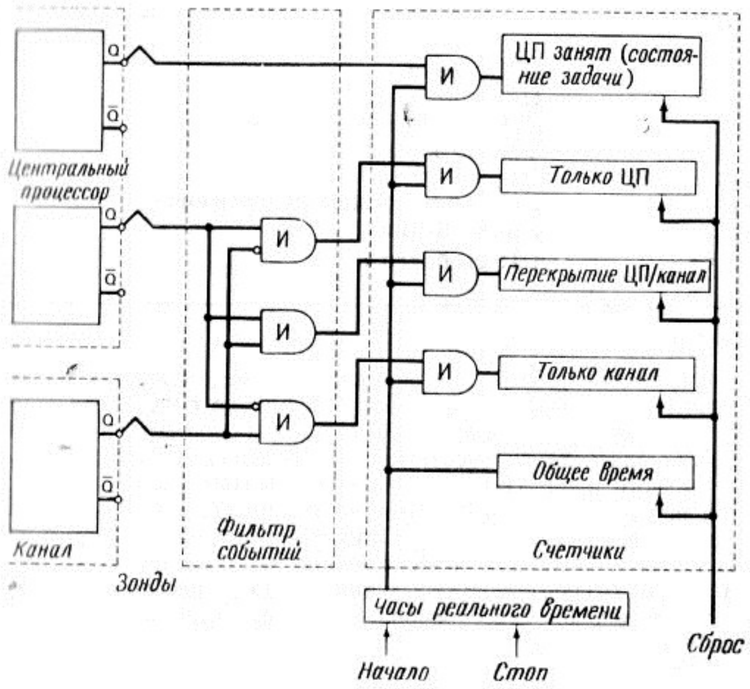


Рис. 2.7. Экспериментальная установка для примера 2.2.

интервала времени, называющегося *сеансом измерения*. Схема измерения иллюстрируется рис. 2.7. Выбор непосредственно измеряемых величин допускает несколько вариантов. Рис. 2.7 показывает, как могут быть измерены первые четыре величины из табл. 2.3 с помощью измерителя с фиксированной программой. Как могут быть получены остальные интересующие нас величины из первых четырех, также показано в этой таблице. Коэффициенты использования состояний пары «процессор — канал» (а именно, ρ_{12}, ρ_{10}, ρ_{20}) можно назвать *коэффициентами перекрытия*.

Основным источником ошибок, искажающих результаты этого эксперимента, называется метод измерения длительностей (см. рис. 2.5). Эта проблема будет обсуждаться в общем виде в разд. 2.4.3. Остальные обычные ошибки могут происходить от неправильного присоединения зонда или от того, что значение наблюдаемого сигнала отличается от приписанного ему экспериментатором. Например, при проведении этого эксперимента крайне важно получить корректные ответы на вопросы типа: когда включается и выключается триггер «задача/супервизор» центральным процессором?

Каково его состояние, когда центральный процессор свободен? В данном примере мы будем предполагать, что центральный процессор всегда находится в состоянии супервизора, когда он свободен.

В схеме рис. 2.7 в результате сжатия объем информации резко сокращается. Если длительность сеанса измерения не слишком велика, разгрузка счетчиков в накопитель не обязательна; в этом случае часть результатов, представляющая интерес для исследователя, доступна наблюдению непосредственно на счетчиках как во время сеанса, так и после него. Элементарные вычисления дают затем остальные величины, приведенные в табл. 2.3.

Таблица 2.3

Результаты эксперимента

Символ	Измеряемая величина	Доля общего времени (в %)
ρ_{10}	ЦП только	13.03
ρ_{12}	Перекрытие ЦП/канал	47.88
ρ_{1p}	ЦП занят (режим задачи)	54.60
ρ_{20}	Только канал	19.04
$\rho_1 = \rho_{10} + \rho_{12}$	ЦП занят	60.91
$\rho_{1s} = \rho_1 - \rho_{1p}$	ЦП занят (режим супервизора)	6.31
$\rho_2 = \rho_{20} + \rho_{12}$	Канал занят	67.92
$\rho_1 = 1 - \rho_1$	ЦП свободен	39.09

Источник: Коленс и Кивиат (1973).

Заметная потеря информации обычно связана с существенным предварительным сжатием. Например, по собранным данным невозможно определить величину перекрытия между каналом и центральным процессором в состоянии задачи. Кроме того, невозможно восстановить длительность каждого отдельного периода активности канала или центрального процессора из-за того, что использован метод накопления на счетчиках. При противоположном подходе мы не проводим никакого предварительного сжатия информации. В этом случае отсутствие предварительного сжатия будет означать запись всех трех кривых, прослеженных зондами. Тогда можно будет ответить на любой вопрос относительно поведения пары «процессор — канал» в деталях, соответствующих имеющейся трассировочной записи, но каждый раз надо будет проводить громадную обработку. ■

2.3.4. Измерители с запоминаемой программой

Развитие измерителей с фиксированной программой привело к тому, что из простых счетчиков они превратились в такие сложные и замысловатые цифровые системы, что замена фиксированной логики набором микропрограмм или программ стала целесообразной и часто удобной модификацией. Таким образом, измерители с запоминаемой программой, которые первоначально использовались только некоторыми изготовителями и в некоторых исследовательских проектах, ныне выпущены в продажу рядом компаний, специализирующихся в области измерений вычислительных систем. Существующие семейства аппаратных измерителей, основанные на миникомпьютерах, предлагают потенциальным потребителям широкий выбор конфигураций, супер-

визорных систем и дополнительных программных и аппаратных возможностей.

Измерители с запоминаемой программой обладают рядом очевидных преимуществ по сравнению с измерителями с фиксированной программой. Главным из них, вероятно, является намного большая гибкость первых, позволяющая им модифицировать схему в ходе эксперимента в зависимости от различных условий. Например, измеритель с запоминаемой программой может быть присоединен к каналу прямого доступа в память измеряемой системы и извлекать любое слово из памяти этой системы; заметим, что это эквивалентно наличию параллельных зондов во всех ячейках памяти при возможности выбора любой из них со скоростью работы электронного оборудования. Кроме того, функции, выполняемые фильтром на схеме рис. 2.3, могут быть изменены во время сеанса измерения в зависимости от частичных результатов или от определенных событий.

Другим преимуществом измерителей с запоминаемой программой является их способность к сжатию данных, не создающему искажений при сжатии данных самой измеряемой системой, как в реальном времени, так и в автономном режиме.

С другой стороны, разрешающая способность этих измерителей имеет тенденцию быть ниже, чем у измерителей с фиксированной программой. Это, вообще говоря, обусловлено дополнительным этапом интерпретации, необходимым для программного управления. Иногда разрешающая способность на самом деле ограничивается характеристиками ввода-вывода обеих систем (измеряемой и измеряющей). Ограничения на ширину полосы пропускания, стандартных соединений ввода-вывода в одной из систем или в обеих можно преодолеть, снабдив измеритель специальным переходником, характеристики которого иногда очень похожи на характеристики измерителей с фиксированной программой. Например, оба коммерческих измерителя с запоминаемой программой, описанные в табл. 2.4, снабжены зондами. Следует также соблюдать осторожность при использовании дополнительных возможностей таких измерителей, так как с их использованием может быть связано значительное количество помех и некоторые нежелательные потери информации.

Следует отметить, что машины, используемые в качестве измерителей с запоминаемыми программами, по характеристикам очень близки к управляющим компьютерам; в данном случае в роли управляемого процесса выступает другой компьютер, который в настоящее время обычно управляется по схеме без обратной связи, хотя некоторые исследования были выполнены с обратной связью (см. разд. 2.5).

Заинтересованный читатель может найти описания и обсуждения конкретных измерителей с запоминаемыми программами у Эстрина и др. (1967), Гринбаума (1969), Роека и Эмерсона (1969), Ашенбреннера и Натараджана (1971) и Хьюза и Кроншоу (1973).

Пример 2.3. Необходимо измерить коэффициент использования некоторых ресурсов программного обеспечения ПРОП-системы. Предположим, что исследова-

Таблица 2.1

Технические характеристики некоторых коммерческих аппаратных измерителей с запоминаемой программой

Изготовитель	COMRESS, Div. of Comten	Tesdata Systems Corp.
Модель	Dynaprobe 8000/II	1200
Зонды		
число	48	144
минимальная длительность импульса (нс)	30	30, 5
максимальн. частота (МГц)	10	15, 40
Логика (коммутационная доска)	600 контактов	1280 контактов
Счетчики		
число	Переменное	16, 4
длина	Переменная	32 бит
максимальн. частота (МГц)	10	20, 40
Длина слова (бит)	16	16
Размер памяти (байт)	64*K*	16*K*
Цикл памяти (мкс/слово)	1	1
Вторичная память	7- или 9-дорожечная лента	Одна или две 7- или 9-дорожечных лент Один 2.5- или 5*M*-байтовый диск (не обязат.)
Периферийные устройства	Телетайп Алфавитно-цифровой дисплей (не обязательно)	Дисплей CRT Поколонная печать (не обязат.)
Программное обеспечение	Общее обеспечение TSM Обеспечение для сжатия данных DYNAPAR/TSM	Пакетный анализатор Дисплей реального времени, диалоговый анализ

тели хотят определить время пребывания этой системы на протяжении определенного сеанса измерений в режиме выполнения операционной системы и в режиме выполнения данного компилятора. Одной из целей этого исследования будет проверка, отвечает ли требованиям имеющееся деление операционной системы на резидентную часть и часть, загружаемую с барабана, или оно могло бы быть улучшено. Таким образом, наблюдения за состоянием триггера «задача/супервизор» (как описано в примере 2.2) недостаточно. Время, в течение которого система находится в состоянии супервизора, должно по крайней мере быть разбито еще на две части: время, когда выполняется резидентная часть операционной системы, и время, когда выполняется часть системы, хранящаяся на барабане.

Резидентная часть находится в фиксированной и привилегированной части памяти. Таким образом, каждый раз, когда центральный процессор находится в состоянии супервизора и программный счетчик указывает на другую область памяти, выпол-

нялся модуль из части, хранящейся на барабане. Очевидно, для обнаружения выполнения этих условий и измерения соответствующей длительности можно использовать измеритель с фиксированной программой, оборудованный устройством параллельного ввода. Однако даже при очень изощренном измерителе с фиксированной программой труднее определить, какой именно модуль, загруженный с барабана, выполняется в данный момент, или измерить коэффициент использования компилятора (который при каждой загрузке в память может занимать новое место). Измеритель с запоминаемой программой, выполненный, например, по схеме рис. 2.8, значительно облегчает эту задачу.

Зонд с последовательным входом соединен с триггером «занят/свободен» процессора барабана. Зонды с параллельными входами присоединены к программному счетчику в центральном процессоре. Программе, работающей в измеряющем компьютере,

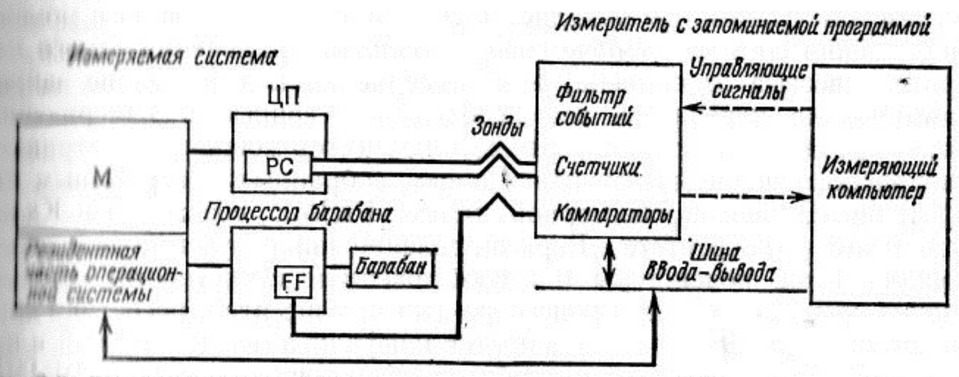


Рис. 2.8. Экспериментальная установка для примера 2.3. PC — программный счетчик измеряемой системы. FF — триггер «занято/свободно» процессора барабана.

известны адреса резидентных таблиц операционной системы, и она может читать их, посылая запросы через шину ввода-вывода в память ПРОП-системы.

Как только обнаружен переход процессора барабана из состояния «свободен» в состояние «занят», измеряющий компьютер получает прерывание. Из таблиц операционной системы и из плана барабана этот компьютер определяет, передается ли блок операционной системы или компилятор. Область, в которую должна загружаться передаваемая информация, также определяется из таблиц. Границы этой области передаются на компараторы, так что когда программный счетчик указывает на адрес, находящийся между этими границами, счетчик, накапливающий время, затраченное на выполнение части операционной системы, хранящейся на барабане, или счетчик, делающий то же самое для компилятора, увеличивается на единицу.

Заметим, что для этого эксперимента требуется по крайней мере три пары компараторов, работающих с содержимым программного счетчика. Одна пара предназначена для резидентной части операционной системы (ее границы никогда не меняются). Другие две пары нужны для обнаружения периодов, когда выполняются модули операционной системы, хранящиеся на барабане, и соответственно для компилятора.

■

Рассматриваемые до сих пор измерители можно назвать *внешними измерителями с запоминаемой программой*, так как они состоят из компьютеров, управляющих измеряемой системой извне. В многопроцессорной системе один из процессоров может быть использован в качестве *внутреннего измерителя с запоминаемой программой* для управления остальной частью системы. Например, аппаратный измеритель, изготовленный для C.mmp — многопроцессорной системы Университета Карнеги — Меллона, содержит несколько регистров, которые

доступны одному из процессоров системы [см. Фуллер и др. (1973)]. Таким образом, этот процессор может управлять экспериментами на другом процессоре и, кроме того, накапливать и сжимать данные, собранные измерителем.

Следует отметить, что внутренний измеритель с запоминаемой программой можно отнести к программным измерителям или точнее к гибридным (см. разд. 2.1), так как он всегда требует добавления к системе набора программ для измеряющего процессора.

Процессор может быть использован в качестве внутреннего измерителя, только если он имеет доступ к областям основной памяти, регистрам или шинам, где появляется собираемая информация в момент обнаружения события. Иначе говоря, область применения такого измерителя весьма ограничена, если измеряемая подсистема не запрограммирована передавать все необходимые данные в измеряющий процессор. В этом случае измеряющий процессор можно рассматривать как приспособление для сжатия данных, собранных программным измерителем. В многопроцессорной системе PRIME, созданной в Калифорнийском университете в Беркли, деление аппаратуры на полностью изолированные подсистемы (по соображениям секретности) сделало невозможным для любого процессора доступ к памяти и регистрам другого процессора. Это один из аспектов конфликта между защитой и измерением. Одной из целей разработки оборудования системы PRIME было соблюдение всех правил защиты системы [см. Феррари (1973)]. Это решение потребовало нескольких жертв, одной из которых был отказ от изготовления внутреннего измерителя с запоминаемой программой.

Внутренний измеритель обнаруживает события посредством непрерывного сравнения содержимого регистров или шин с заранее определенными комбинациями битов или посредством слежения за сигналами, приходящими от измеряемой подсистемы или от своих собственных часов. Данные могут быть собраны через каналы межпроцессорной связи. В этом случае полоса пропускания этих каналов должна позволять дополнительную нагрузку без чрезмерного снижения производительности измеряемой подсистемы. Эти же требования должны быть выполнены при связи процессоров через основную память (вместо каналов передачи данных). Для накопления результатов измерений следует применять внешние накопители, которые используются измеряемой подсистемой не слишком интенсивно.

2.4. ПРОГРАММНЫЕ И МИКРОПРОГРАММНЫЕ ИЗМЕРИТЕЛИ

2.4.1. Введение

В разд. 2.2 *программные измерители* были определены как измерители, состоящие из команд, добавляемых к системе (к аппаратуре и программному обеспечению) для сбора данных, относящихся к ее производительности. То, что эти дополнительные команды должна вы-

полнять сама измеряемая система, приводит к возникновению искажений. Количество искажений зависит от частоты обнаруживаемых событий и от операций, выполняемых измерителем при обнаружении каждого события. Обычно можно идентифицировать два компонента искажений: *временной и пространственный*. Пространство и время часто могут заменять друг друга: пространство можно уменьшить (за счет увеличения объема предварительной обработки) при дополнительном расходе времени и наоборот.

Например, если надо определить среднее и стандартное отклонение некоторой величины по ряду последовательных измерений этой величины, минимальные искажения по времени получатся, если накопить измеренные значения и отложить вычисление среднего и стандартного отклонения на последующее время. В противоположном случае мы могли бы хранить оценки среднего и стандартного отклонения на протяжении эксперимента, обновляя их с каждым новым измеренным значением [см. (3.7) и (3.8) в примере 3.2]. Ясно, что второй способ вносит больше временных искажений, чем первый за счет обновления оценок. Однако пространственные искажения в этом случае меньше, так как требуется лишь пространство для двух оценок, нового значения измеряемой величины и для программы обновления.

Иногда ошибки, возникшие из-за искажений, можно полностью или частично корректировать. Например, время, затраченное в ходе эксперимента на сбор и предварительное сжатие данных, можно измерить и вычесть из длительностей интервалов, искаженных им. Воздействие пространственных искажений на измеряемые величины обычно определить намного труднее.

Разрешающая способность программных измерителей обычно ниже, чем аппаратных. Программные измерители могут обнаруживать только более макроскопические, менее частые события. Их область применения отличается от области применения аппаратных измерителей, хотя и существуют некоторые типы событий, которые могут быть обнаружены обоими. Если одна и та же величина может измеряться аппаратным и программным измерителями, данные аппаратного измерителя считаются более точными из-за меньших искажений [см., например, Карлсон (1971) и Питерсон (1974)]. Таким образом, аппаратные измерители можно использовать для контроля точности некоторых программных измерителей.

Программный измеритель может быть более гибким, чем аппаратный в своей области применения. Покупные цены коммерческих программных измерителей имеют немного более узкий спектр, сосредоточенный вблизи области минимальных цен для аппаратных измерителей. Их несколько легче вводить в действие, но для этой операции часто требуется квалифицированный персонал. Для пользования ими и интерпретации выходных данных также необходима некоторая подготовка.

Микропрограммные измерители по своей природе и особенностям подобны программным, но соотношение пространственных и времен-

ных искажений у них имеет другие характеристики. Вообще, пространство — более ограниченный ресурс для микропрограмм, чем для программ, а время выполнения микрокоманды меньше. Однако время для микропрограммного обеспечения иногда может оказаться критическим фактором, что редко имеет место для программного обеспечения. Кроме того, микропрограммные измерители имеют более высокую разрешающую способность, но их возможности предварительной обработки гораздо более сильно ограничены искажениями. По области применения микропрограммный измеритель (в терминах обнаружимых событий и измеримых величин) занимает промежуточное положение между аппаратным измерителем и программным. Гибкость микропрограммных измерителей зависит от легкости модификации управляющей памяти.

Ряд микропрограммных измерителей, включенных в систему IC 7000 фирмы Standard Computer Corporation, снабженных записываемой управляющей памятью как в центральном процессоре, так и в процессоре ввода-вывода, описан Саалом и Шустеком (1972). Среди прочих величин, эти измерители могут измерять коэффициенты использования областей основной памяти данной программой и коэффициенты использования кодов операций (и пар кодов операций) центрального процессора. Они могут также вести трассировку программы. Для того чтобы этим воспользоваться, необходимы привилегированная буферная область в основной памяти и лентопротяжное устройство. Они существенно снижают скорость выполнения, особенно из-за высокой частоты сбора данных, из-за использования магнитной ленты в качестве накопителя, а также из-за конфликтов между измерением ввода-вывода и нормальным вводом-выводом в процессоре ввода-вывода.

Существуют программные измерители, снабженные микропрограммными дополнениями к измеряемой системе. Например, в микропрограммном процессоре в набор команд могут быть включены посредством ввода в управляющую память соответствующих микропрограмм команды центрального процессора, ориентированные на измерение. Хотя нет необходимости делать машину микропрограммной, поскольку команды, предназначенные для измерительных целей, могут быть встроены в процессор аппаратно, микропрограммирование позволяет добавлять такие команды на более поздних этапах.

В качестве простейшего примера на рис. 2.9 приведен формат команды для записи событий для 32-битовой машины с 8-битовым кодом операции, смоделированный по формату RX Системы IBM/360 и 370. Команда считывает значение часов и запоминает это значение и содержимое поля E (идентификатор события) в памяти по адресу (B)+(X)+D, где (B) означает содержимое регистра B, (X) — содержимое регистра X, а D — смещение. И, наконец, команда увеличивает содержимое X.

Другой пример, который более соответствует понятию микропрограммного измерителя, поскольку состоит из единственной микро-

команды, можно взять из Burroughs B1700 [Вилнер (1972)]. В этой машине микрокоманда «монитор» выдает определенный программистом набор битов на точки схемы, доступные зондам аппаратного измерителя. Это также пример *гибридного измерителя*. Гибридные измерители состоят из внешних аппаратных измерителей, получающих данные,

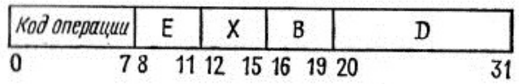


Рис. 2.9. Формат простой команды, ориентированной на измерения.

собранные для них программным или микропрограммным измерителем (работающим в измеряемой системе). Свободова (1973) описывает конструкцию гибридного измерителя для Системы IBM/370 модель 165. Другие гибридные измерители описаны Эстрином и др. (1967) и Хьюзом и Кроншоу (1973).

Аппаратные, программные и микропрограммные измерители в некотором смысле дополняют друг друга, так как существуют события и данные, которые можно регистрировать или к которым можно (или экономичней) получить доступ только с помощью одного из этих типов измерителей. Это наблюдение наводит на мысль, что наиболее удобным решением был бы комплексный аппаратно-программно-микропрограммный подход к измерениям. Координация между внешним оборудованием и программным измерителем внутри системы, существующая в гибридных измерителях, является примером тенденции к объединению различных методов. Однако совместное использование измерителей одного или разных типов, их координация и разделение функций выдвигают все еще открытые проблемы. Вместо того чтобы снабжать систему измерительными средствами после разработки и изготовления, следовало бы включать в нее достаточный набор их еще на стадии проектирования. Взаимоотношения между этими средствами и между ними и остальной частью системы должны быть определены во время проектирования системы. Эта практика пока еще не очень популярна среди изготовителей машин, но принята для некоторых исследовательских машин, таких, как Multics [Зальцер и Джинтелл (1970)] и PRIME [Феррари (1973)]. Некоторые из проблем, возникающих при конструировании измерительных систем, обсуждаются в разд. 2.5.

2.4.2. Внутренне управляемые измерители

Программные методы позволяют нам обнаруживать внутренние события следующих видов: получен доступ к переменной или структуре; изменено значение переменной; переменная принимает данное значение; выполняется заданная команда или команда заданного типа. Большая часть этих событий может быть обнаружена прямо или косвенно измерителями, обнаруживающими выполнение заданной команды. Например, все обращения к определенному хранящемуся объекту

можно обнаружить по выполнению всех команд, адресующихся к этому объекту.

В некоторых случаях удобнее использовать другие методы обнаружения. Например, если нас интересует подсчет всех обращений к данной структуре данных, было бы более естественно следить за «точкой входа» этой структуры данных, чем за выполнением всех команд, дающих к ней доступ. Однако это, вообще говоря, возможно только в том случае, если доступ к данным осуществляется через единственный дескриптор или подпрограмму, в которые можно вставить измеритель. Возможным, но с экономической точки зрения сомнительным решением было бы использовать один бит каждого элемента данных или команды в качестве индикатора ловушки. Каждый раз, когда происходит обращение к некоторому слову памяти, ловушка генерируется аппаратно или микропрограммно, если этот бит равен 1. Это дополнение можно использовать как для измерений, так и для отладки программ — деятельности, которая имеет много общего с измерением программного обеспечения.

Таким образом, единственным типом внутреннего события, реально обнаружимым внутренне управляемым программным измерителем, является выполнение команды. Датчик, называемый *контрольной точкой*, или *программным зондом*, очень похож на точку прерывания в диалоговой отладке. Однако после выполнения своей задачи контрольная точка передает управление не пользователю, а измеряемой программе. Контрольная точка есть команда, вставленная в нужное место программы. Она или собирает и накапливает данные сама, как в случае команды на рис. 2.9, или вызывает *измерительную программу* для той же цели.

Ошибки в отсчете времени, обусловленные контрольной точкой, равны времени выполнения контрольной точки и измерительной программы, если она есть. Это время часто можно установить довольно точно; иногда, если система снабжена часами с высокой разрешающей способностью и накладные расходы на чтение часов не слишком велики, его можно измерить. В этих случаях ошибки измерения за счет временных искажений можно компенсировать или существенно уменьшить, корректируя записанные времена событий. Оценка воздействия временных искажений на другие типы данных обычно сложнее. Эффект пространственных искажений, обусловленных потребностью в пространстве для дополнительных кодов (контрольные точки и измерительные программы) и для собранных данных, определить также весьма трудно. Заметим, что искажения во времени могут быть недопустимы, если программные зонды стоят в критических по времени частях операционной системы, например в управляющих синхронными устройствами, такими, как диски или барабаны.

Пример 2.4. Измеряется распределение команд, выданных пользователями в диалоговой системе. Интерпретатор команд — модуль операционной системы, активируемый каждый раз, когда команда поступает с терминала. Этот модуль интерпретирует команду и передает управление той своей части, которая обрабатывает эту команду. Предполагая, что надо получить относительные частоты для всех типов команд, сопоставим типы команд элементам вектора COMTYP и будем хранить в каждом элементе количество поступлений соответствующей команды. В начале сеанса измерения вектор будет заполнен нулями с помощью некоторой привилегированной команды, и экспериментатору будет разрешено в любое время читать этот вектор с помощью другой привилегированной команды. Контрольные точки, показанные в табл. 2.5 (a), выполняют задачу накопления гистограммы использования команд в COMTYP. Язык, использованный в таблице,— ФОРТРАН. Каждый раздел программы, обрабатывающий отдельный тип команды, начинается с контрольной точки, увеличивающей соответствующий элемент вектора COMTYP.

Если необходимо накапливать трассировку команд, можно воспользоваться средствами, приведенными в табл. 2.5 (b). Запись о каждом событии состоит из метки события (типа команды) и времени события. Метки накапливаются в векторе EVTAG, а времена — в векторе EVTIME. STORE — имя программы, не приведенной в табл. 2.5, которую вызывают, когда эти два вектора заполнены, и которая переписывает их содержимое во внешнюю память. ITIME — тоже имя программы, не показанной в таблице, которая считывает показания часов реального времени и превращает их в целое число K.

Очевидно, что решение в табл. 2.5 (b) вызывает существенно большие искажения, чем в табл. 2.5 (a), как по времени, так и по пространству. Однако трассировка команд содержит намного больше информации, чем гистограмма времен выполнения команд. Более того, природа измеряемых величин такова, что эти искажения не влияют на

Таблица 2.5

Программные измерители для задачи в примере 2.4

(a)

```
C  ИНТЕРПРЕТАТОР КОМАНД
                .
                .
                .
C  РЕДАКТИРОВАНИЕ БОК
100 COMTYP(1) = COMTYP(1) + 1
                .
                .
                .
C  КОМПИЛЯЦИЯ БОК
200 COMTYP(2) = COMTYP(2) + 1
                .
                .
                .
C  ЗАПОМИНАНИЕ БОК
300 COMTYP(3) = COMTYP(3) + 1
                .
                .
                .
```

(b)

```
C  ИНТЕРПРЕТАТОР КОМАНД
                .
                .
                .
C  РЕДАКТИРОВАНИЕ БОК
100 CALL MEASUR(1)
                .
                .
                .
C  КОМПИЛЯЦИЯ БОК
200 CALL MEASUR(2)
                .
                .
                .
C  ЗАПОМИНАНИЕ БОК
300 CALL MEASUR(3)
                .
                .
                .
    SUBROUTINE MEASUR(COMD)
    COMMON NCOMD
    DIMENSION EVTAG(1000),
      EVTIME(1000)
    CALL ITIME(K)
    EVTAG(NCOMD) = COMD
    EVTIME(NCOMD) = K
    NCOMD = NCOMD + 1
    IF(NCOMD.LE.1000) RETURN
    CALL STORE
    NCOMD = 1
    RETURN
    END
```

(c)

```
    SUBROUTINE MEASUR(COMD)
    COMMON NCOMD,TOTERR
    DIMENSION EVTAG(1000),
      EVTIME(1000)
    CALL ITIME(K)
    EVTAG(NCOMD) = COMD
    EVTIME(NCOMD) = K - TOTERR
    NCOMD = NCOMD + 1
    IF(NCOMD.LE.1000) GO TO 10
    CALL STORE
    NCOMD = 1
10  CALL ITIME(K1)
    TOTERR = TOTERR + K1 - K
    RETURN
    END
```

(Здесь БОК — блок обработки команд.)

точность измерений в случае табл. 2.5 (a), состоящих лишь в подсчете событий. В случае табл. 2.5 (b) искажения могут привести к тому, что записанные времена выполнения команд будут несколько неточными по сравнению с теми, которые были бы измерены аппаратным измерителем. Однако точность, желательная для времен выполнения команд, вероятно, достаточно низка, так что эти ошибки допустимы для всех возможных применений времен выполнения команд в исследовании по оценке производительности.

В случае необходимости можно было бы измерить время, затраченное на выполнение программы MEASUR, и автоматически скорректировать времена команд, как показано в табл. 2.5 (c). Заметим, что в начале сеанса измерения общей ошибке TOTERR присваивается значение, равное нулю. Разумеется, разрешающая способность часов должна бы быть достаточно высокой, чтобы это измерение проводилось с разумной точностью. Например, часы с шагом по времени, равным 1 микросекунде, удовлетворяют этому условию, тогда как с шагом 1 миллисекунда — нет. ■

Контрольные точки можно использовать как для фиксированных, так и для нефиксированных программных измерителей. *Фиксированный программный измеритель* это постоянная контрольная точка, собирающая данные каждый раз, когда она получает управление. В операционных системах можно найти много примеров фиксированных измерителей.

Очень распространено использование фиксированных измерителей для целей учета. Типичная программа учета активируется, когда какое-нибудь задание покидает систему, читает различные регистры и таблицы, вычисляет по этим данным ресурсы, использованные заданием, и, используя алгоритм вычисления цен на данной установке, выписывает счет пользователю. Статистические данные, собранные программой учета, обычно накапливаются в специальном программно реализуемом журнале и периодически обрабатываются, чтобы персонал установки мог следить за производительностью системы хотя бы с заметным запаздыванием. Этот журнал часто служит ценным источником данных о производительности системы для работ по оценке. Некоторые учетные программы, созданные с ориентировкой на измерение, выдают отчеты, показывающие значения таких индексов производительности, как пропускная способность и коэффициенты использования оборудования.

Иногда в системе ведутся другие типы журналов, например журнал обнаруженных ошибок, сбоев и простоев различных модулей; журнал попыток нарушения секретности файлов пользователя; журнал характеристик заданий на существенно более детальном уровне, чем записи в журнале учета. Журнал обычно требует большого количества постоянной памяти и накапливается на диске или ленте.

Программометр — другой полезный тип фиксированного измерителя. Он состоит из постоянной контрольной точки, единственная функция которой состоит в том, чтобы увеличивать на 1 содержимое некоторой ячейки памяти каждый раз, когда выполняется контрольная точка. Эта ячейка доступна программе, когда понадобится, и последовательные считывания ее содержимого могут обеспечить подсчет событий, нужный для оценки или непрерывного наблюдения. Например, программометры, подсчитывающие количество выполнений каж-

дого модуля операционной системы, являются ценной составной частью системы. Другим примером является программометр, подсчитывающий прерывания центрального процессора. Разработчики системы Multics ввели в свою систему ряд программометров и других чуть более сложных фиксированных измерителей. Эти измерители оказались очень полезными, особенно из-за трудности и риска введения временных измерителей в большую программную систему [см. Зальцер и Джинтелл (1970)].

Более замысловатый тип фиксированного измерителя должен быть использован для измерения длительности интервалов между событиями. Например, часы центрального процессора в мультипрограммной системе считывают часы реального времени в моменты начала и окончания использования центрального процессора некоторым процессором и накапливают разности между последовательными отсчетами.

Гибкость *нефиксированных измерителей* может варьироваться в широких пределах. Некоторые из них можно только включать или выключать, а обнаруживаемые ими события нельзя изменить. Так обстоит дело для большей части измерителей, встроенных в систему Multics. У других измерителей, таких, как Программа внутренней оценки производительности систем (SIPE) [Денистон (1969)], которая может быть встроена в комплекс резидентных модулей операционной системы TSS/360, пользователь может задать подмножества событий из наблюдаемых этим измерителем, которые необходимо записать. Другие, как, например, средства управления системой (SMF) фирмы IBM [см. Помрой (1972)], позволяют пользователям писать свои собственные программы измерения и сжатия данных, но накладывают ограничения на размещение контрольных точек. SMF, в частности, собирает информацию о ресурсах, используемых каждым заданием на каждом шаге, и обеспечивает несколько *выходов из управляющей программы*, причем эти выходы могут использоваться для вызова дополнительных программ обработки, предназначенных специально для данной установки. Эти выходы соответствуют важнейшим этапам прохождения задания внутри системы, которые можно обнаружить и записать с помощью программ, добавленных пользователями системы SMF.

Два программных измерителя предоставляют почти полную свободу своим пользователям, это Информатор и SMT. Они могут применяться для измерения любой программы, а не только операционной системы. Более того, они используют возможности диалоговых систем по временному измерению программ, которое часто оказывается итеративным (метод проб и ошибок).

Информатор [Дойч и Грант (1971)] представляет собой пакет программ, разработанный и внедренный для системы разделения времени Xerox 940. Он автоматически расставляет контрольные точки в теле данной программы в местах, определенных пользователем. Контрольные точки могут вызывать измерительные программы, подготовленные пользователем и проверенные Информатором на выполнение некоторых критериев приемлемости. Эти критерии будут удовлетворены, если из-

мерительная программа, размещенная в памяти измеряемой программы,

a) не осуществляет запись на внешний накопитель или передачи управления,
b) не содержит ошибочные или привилегированные команды,
c) выполняется только в течение заданного времени,
d) не модифицирует себя или другие измерительные программы,
e) не имеет возможности повторного входа без выхода из подпрограммы,
f) не считывает защищенные разделы программ или данных.

Для облегчения отладки измерительных программ все они должны иметь стандартную структуру, состоящую из нескольких различных областей с различными характеристиками доступа. Кроме того, запрещено использование передач управления назад, за исключением случая, когда передача управления замыкает цикл, который должен выполняться не более чем постоянное заранее известное число раз. Это ограничение позволяет Информатору быстро вычислять оценку наихудшего случая времени выполнения измерительной программы. Обращения к операционной системе для измерительной программы незаконны, кроме определенных операций ввода-вывода, связанных с

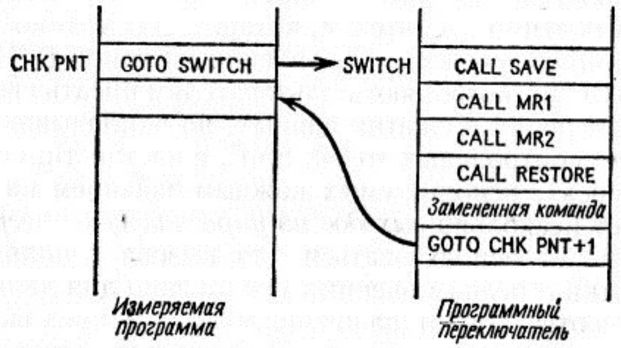


Рис. 2.10. Программный переключатель, вызывающий подпрограммы MR1 и MR2 (переключатель можно выключить, заменив CALL MR1 и GALL MR2 пустыми командами).

параллельным использованием аппаратного измерителя. Чтобы контролировать выполнение условия «е», Информатору передается таблица защищенных областей программы. Так как в некоторых областях адресного пространства программы нельзя помещать контрольные точки (области данных, команды «выполнить», области команд, время исполнения которых критично), Информатору должна быть предоставлена также таблица таких областей. Кроме того, Информатору нужны таблицы максимально допустимых искажений по времени для каждой измеряемой области программы и таблица свободных областей адресного пространства для размещения в них самих измерительных программ.

Информатор использует *метод заплат* для установки контрольных точек (рис. 2.10). Обращение к измерительной программе (или программам) производится через *программный переключатель*, размещенный в одной из областей пространства измерительной программы. Переключатель, конструкция которого ясна из рис. 2.10, может быть включен для связи контрольной точки с измерительной программой или выключен для разрыва этой связи. Если сбор некоторых данных не нужен, соответствующие переключатели можно выключить для уменьшения степени снижения производительности за счет процесса измерения.

Пользователи могут взаимодействовать со своими измерительными программами, что особенно полезно при отладке этих программ. Диалоговое взаимодействие обеспечивается частью Информатора, размещенной постоянно в адресном пространстве измеряемой программы. Это позволяет выполнять некоторые проверки, которые возможны только во время выполнения программы, но увеличивает искажения от измерителя. Элементарные команды, обеспечивающие общение между Информатором и его пользователями, перечислены в табл. 2.6.

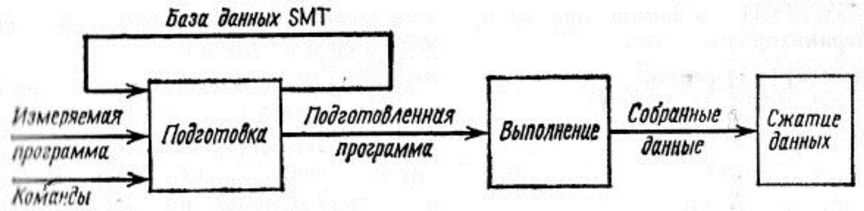


Рис. 2.11. Три этапа работы измерителя SMT. (*Источник:* Феррари и Лиу (1975), с разрешения John Wiley & Sons Ltd.)

Программный измеритель (SMT) представляет собой пакет программ (имеющий ту же область применения, что Информатор), разработанный для измерения внутренних процессов в системе PRIME [см. Феррари и Лиу (1975)]. Три этапа работы SMT схематически изображены на рис. 2.11. Измеритель SMT был задуман как универсальное измерительное средство для всех пользователей системы PRIME. Поскольку в системе PRIME, максимально отвечающей требованиям секретности, пользователи не могут измерять ничего вне дозволенных пределов, большая часть ограничений, накладываемых Информатором на размещение контрольных точек и на измерительные программы, оказывается необязательной. Что бы ни делали пользователи в пределах санкционированного доступа, это не отражается на остальной части системы. Операционная система полностью защищена от несанкционированного вмешательства пользователей и может быть измерена только немногими системными программистами, имеющими соответствующее разрешение. Этим программистам разрешено добавлять к своей копии SMT некоторые автоматические проверки измерительных программ с целью защиты операционной системы от их непреднамеренных ошибок.

Таблица 2.6

Набор основных команд для Информатора

Команда	Действие
NEW MEASUREMENT PROGRAM (<адрес в пространстве пользователя>)	Проверяет подпрограмму в пространстве адресов пользователя и, если она корректна, копирует ее в пространство адресов операционной системы
ATTACH (<идентификатор>), (<адрес в операц. системе>)	Вставляет по указанному адресу контрольную точку, вызывающую измерительную программу с данным идентификатором
DELETE MEASUREMENT ROUTINE (<идентификатор>)	Удаляет указанную программу и отключает все контрольные точки, вызывающие ее
DETACH CHECKPOINT (<адрес в операц. сист.>)	Отключает указанные контрольные точки от их переключателей
SET SWITCH (<номер перекл.>, <идентификатор>)	Связывает данную программу с указанным переключателем
CLEAR SWITCH (<номер перекл.>)	Разрушает связь указанного переключателя со всеми программами
BLOCK ( )	Вызывает паузу в выполнении программы пользователя до тех пор, пока одна из измерительных программ не активирует ее
EXECUTE IN CONTEXT (<команда>)	Выполняет команду так, как будто бы она входит в состав измерительной программы
ACCESS MEASUREMENT SPACE (<адрес в простр. пользователя>)	Помещает рабочее поле измерительной программы в виртуальное пространство пользователя с разрешением только считывать информацию
IS CHECKPOINT? (<адрес в операц. системе>)	Сообщает пользователю, есть ли контрольная точка по указанному адресу
IS CONNECTED? (идентификатор)	Выдает список контрольных точек, с которыми связана указанная программа

Источник: Дойч и Грант (1971).

Измеритель SMT не только обслуживает программы, написанные пользователями, но и предлагает пользователям ряд стандартных измерительных программ (см. их типы в табл. 2.7). Стандартные программы уменьшают искажения (так как их рабочие характеристики тщательно оптимизированы) и облегчают пользование измерителем.

Измерительные программы могут быть написаны на любом языке, для которого есть компилятор в системе PRIME. Кроме того, каждой программе может быть дано несколько имен, так что пользователь мо-

Таблица 2.7

Типы стандартных измерительных программ в системе SMT

Измерительная программа	Выполняемые действия
CT (<номер счетчика>)	Увеличивает содержимое одного из 16 счетчиков
UP (<имена регистров>)	Записывает содержимое программного счетчика и (по желанию) содержимое регистров A и (или) B и (или) X
UV (<имя переменной>)	Записывает значение переменной
UP (<имя переменной>)	Записывает значение переменной и содержимое программного счетчика
IT (<предел>), <флаг завершения программы>)	Останавливает сбор данных и выключает контрольные точки, как только число выполнений программы достигает предела; останавливает программу, если установлен флаг

Источник: Феррари и Лиу (1975).

жет по выбору отключать или полностью уничтожать некоторые обращения к программе, оставляя остальные нетронутыми. Расположение контрольных точек можно указать символическими адресами в формате, используемом для диалогового отладчика системы. В измерителе SMT есть макрокоманды, например, трассировки передач управления внутри некоторой программы. Конструкция измерителя включает также средства определения макрокоманд, средства подключения программ, обеспечивающие вызов данной стандартной программы или программы пользователя в каждый исследуемый момент, и расширяемую библиотеку стандартных программ сжатия данных.

Пример 2.5. Использование SMT для исследования генератора псевдослучайных чисел (RAND) иллюстрируется табл. 2.8 и 2.9. Команды измерителя SMT, использо-

Таблица 2.8

Программа генерации псевдослучайных чисел, которая должна быть исследована в примере 2.5

RAND	BSS	1	запоминание адреса возврата
	LDA	VALUE	загрузка старого значения в регистр A
	MUL	MFAC	умножение на MFAC; старшие биты в регистре A, младшие биты в регистре B
	CBA		копирование регистра B в регистр A
	ADD	AFAC	сложение AFAC с регистром A
	STA	VALUE	запоминание нового значения = (старое значение × MFAC + AFAC) (mod 2^{24})
	BRR	RAND	возврат с новым значением в регистре A

Источник: Феррари и Лиу (1975), с разрешения John Wiley & Sons Ltd.

ванные в табл. 2.9, перечислены в табл. 2.10. В табл. 2.8 приведен текст исследуемой программы. Программа RAND написана на языке ассемблера системы PRIME, совпадающем с языком системы Xerox 940. Производится трассировка и накапливается трассирующая запись первых чисел, генерируемых программой RAND. В табл. 2.9 (a) показаны команды, которые должен ввести пользователь (прописные буквы), и ответы системы (строчные буквы). Файл /PROG содержит генератор RAND и программу, использующую его, файл /INSPROG создается для запоминания измеряемых программ и связанной с ними базы данных SMT. Две стандартные измерительные программы загружены при помощи SMT в адресное пространство RAND (см. табл. 2.7). Одна из них, названная пользователем RANVAL, относится к типу программ, читающих значение переменной (RV); она записывает значение VALUE и активизируется одной из контрольных точек для выполнения непосредственно перед возвратом BRR RAND. Другая измерительная программа принадлежит к типу завершающих сбор данных (TC) и называется TCNT; она завершает сбор данных, не прекращая выполнения программы, после того, как отсчитывается 1000 ее активизаций. Программа TCNT вызывается из того же места памяти, что и RANVAL, и включает все контрольные точки, если удовлетворены условия завершения.

Таблица 2.9

Сеанс работы SMT по подготовке программы из табл. 2.8

```
(a) @SMT.
    # LOAD. file: /PROG.
    # DEFINE. mrname: RANVAL
          type: RV
          loc: VALUE
    # DEFINE. mrname: TCNT
          type: TC
          limit: 1000
          halt?: N
    # CHECKPOINT. mrname: RANVAL
          loc: RAND+6
    # CHECKPOINT. mrname: TCNT
          loc: RAND+6
    # SAVE. file: /INSPROG
          starting address: GO
    # FINISH.
    @GO TO /INSPROG.
    data file: /smtdtØ

(b) @SMT.
    # RELOAD. file: /INSPROG
    # ON. all?: Y
    # FORGET. all?: N
          c:
          m: RANVAL
    # DEFINE. mrname: CALLER
          type: RV
          loc: RAND
    # CHECKPOINT. mrname: CALLER
          loc: RAND+1
    # SAVE. file: /INSPROG
          starting address: GO
    # FINISH.
    @GO TO /INSPROG.
    data file: /smtdt1
    @
```

Источник: Феррари и Лиу (1975), с разрешения John Wiley & Sons Ltd.

Обеспечив запоминание в машине подготовленной к измерению программы, пользователь тем самым завершает этап подготовки программы с помощью SMT. Далее запоминается подготовленная программа. Результаты измерений накапливаются в файле /SMTDT0. Имя этого файла выбирает SMT.

В табл. 2.9 (b) показаны команды, которые должен ввести пользователь для удаления измерительной программы RANVAL и, следовательно, вызывающей ее контрольной точки. Кроме того, добавлена другая стандартная измерительная программа, названная CALLER с контрольной точкой, помещенной между первым и вторым операторами программы RAND. Программа CALLER относится к типу RV и, будучи активизирована, записывает адрес возврата. Таким образом, файл /SMTDT1 после выполнения измеряемой программы будет содержать адреса возврата для первых 1000 вызовов программы RAND. Эти два файла данных могут быть затем сжаты так, чтобы пользователь системы мог легко понять и интерпретировать результаты проведенных двух экспериментов.

Таблица 2.10

Команды SMT, использованные в табл. 2.9 (см. пример 2.5)

Команда	Аргументы	Действие
LOAD	<имя файла>	Загружает и связывает двоичные файлы, содержащие измеряемую программу
RELOAD	<имя файла>	Загружает файл — образ памяти, созданный на предыдущей стадии подготовки, и связанную с ним базу данных SMT
DEFINE	<имя измер. прогр.>	Присваивает имя (и аргумент) стандартной или написанной пользователем программе
CHECKPOINT	<имя измер. прогр.>, <адрес>	Вставляет вызов указанной программы в назначенное место измеряемой программы
ON	<флаг>, <адрес>, <имя измер. прогр.>	Включает заданную контрольную точку
FORGET	<флаг>, <адрес>, <имя измер. прогр.>	Удаляет заданные контрольные точки
SAVE	<имя файла>	Записывает в заданный файл образ памяти исследуемой программы и ее базы данных SMT
FINISH		Завершает фазу подготовки

Источник: Феррари и Лиу (1975).

Другие примеры использования метода контрольных точек при измерении программ приведены в разд. 9.3.1.

2.4.3. Внешне управляемые (выборочные) измерители

Методы выборочного исследования позволяют существенно уменьшить количество данных, которые необходимо собрать для оценки некоторых интересующих нас величин. Они могут основываться на об-

наружении внутренних событий. Например, данные могут собираться каждый раз, когда содержимое некоторого счетчика достигает определенного значения. Этот способ можно назвать *выборка по счетчику*. Примерами таких измерителей служат измерители, собирающие данные после выполнения заданного числа команд или после заданного числа обращений к памяти.

В большинстве случаев, однако, выборочные измерители обнаруживают наступление заданных моментов времени, названных в разд. 2.2 *внешними событиями*. Этот тип измерителя можно назвать измерителем с *выборкой по времени*. Внешне управляемые измерители обычно относятся к типу измерителей с выборкой по времени.

Моменты выборки могут разделяться интервалами постоянной длительности *(периодическая выборка)* или случайными *(случайная выборка)* или интервалами, подчиненными каким-то еще законам изменения. Когда событие обнаружено, выборочный измеритель действует как внутренне управляемый, собирая (а, возможно, обрабатывая и накапливая) данные, относящиеся к данному эксперименту. Данные, собранные выборочным измерителем, называются *наблюдениями*.

Выборочный метод может быть использован в аппаратных измерителях. Например, измерение коэффициентов использования, показанное на рис. 2.5 и 2.7, основано на периодической выборке электрических сигналов. Однако эта выборка осуществляется внутри измерителя, непрерывно получающего наблюдаемые сигналы. Таким образом,

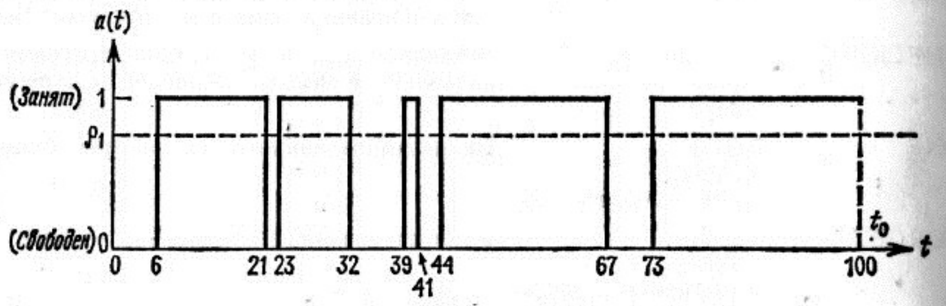


Рис. 2.12. Состояние центрального процессора как функция времени.

изменения напряжения этих сигналов можно использовать, как и во внутренне управляемых измерителях, например их можно подсчитывать. Если мы исключим этот тип выборки из рассмотрения, можно сказать, что выборочные методы более широко применяются для программных измерителей, чем для аппаратных. Поэтому они и обсуждаются здесь.

Пусть нужно измерить коэффициент использования центрального процессора. Тогда, как показано на рис. 2.5, мы можем представить состояние центрального процессора булевой функцией $[a(t)$ на рис. 2.12]. В каждый выборочный момент t_i мы наблюдаем и записываем значение $a_i = a(t_i)$. Последовательность $a_1, a_2, a_3, \ldots, a_i, \ldots$ можно

моделировать как стохастическую последовательность, т. е. как дискретный по времени стохастический процесс.

Стохастический процесс $X(t)$ есть функция времени t, значения которой суть случайные переменные. Значение $X(t)$ в момент t^* представляет собой состояние стохастического процесса в момент t^*. Если каждая случайная переменная может принимать лишь конечное или счетное число значений, мы имеем *процесс с дискретными состояниями*, или *цепь*. В противном случае мы имеем *процесс с непрерывными состояниями*. Для идентификации состояний цепи $X(t)$ можно использовать натуральные числа. Если $X(t)$ находится в состоянии S_j в момент t, мы говорим, что $X(t) = j$, где j — натуральное число. Если моменты времени, в которые возможны переходы, образуют конечное или счетное множество, мы имеем *процесс с дискретным временем*, или стохастическую последовательность. Эти моменты времени можно заменить целыми значениями индекса времени i. Следовательно, стохастическую последовательность можно обозначать X_i $(i = 1, 2, \ldots)$. Конечная стохастическая последовательность называется также *временным рядом*. Если переходы могут происходить в любой точке непрерывной оси времени, то процесс относится к *процессам с непрерывным временем*.

Возвращаясь к нашей задаче, мы видим, что если нас интересует коэффициент использования центрального процессора ρ_1 на протяжении интервала $(0, t_0)$, нам следует измерять среднее по времени $\bar{a}$ от $a(t)$ на этом интервале, поскольку $\rho_1 = \bar{a}$. Если проведено n измерений величины $a(t)$ на интервале $(0, t_0)$, мы получим дискретный временной ряд $a_1, \ldots, a_n$. Оценкой ρ_1 будет *выборочное среднее*

$$\hat{\rho}_1 = \frac{1}{n} \sum_{i=1}^{n} a_i. \tag{2.2}$$

Наш временной ряд *независим*, если случайные переменные $a_1, a_2, \ldots, a_n$ статистически независимы друг от друга. Обоснование выборочного метода существенно опирается на предположение о независимости наблюдений.

Примем это предположение, и пусть σ_1 — стандартное отклонение распределения значений $a(t)$ в интервале $(0, t_0)$. Тогда по центральной предельной теореме [см., например, Снедекор и Кокрен (1967)] распределение величины $(\hat{\rho}_1 - \rho_1)/(\sigma_1/\sqrt{n})$ стремится к стандартному нормальному распределению при стремлении n к бесконечности. Стандартное нормальное распределение есть нормальное распределение со средним 0 и дисперсией 1. Таким образом, $\hat{\rho}_1$ стремится к нормально распределенной величине со средним ρ_1 и дисперсией σ_1^2/n при увеличении n.

Если наблюдения коррелированы друг с другом, среднее распределения $\hat{\rho}_1$ по-прежнему стремится к ρ_1, но дисперсия уже отличается от σ_1^2/n. Общее выражение дисперсии $\hat{\rho}_1$ через ρ_1 дано в разд. 2.6.4.

Если существует некоторая функциональная связь между наблюдениями, то выборочное среднее $\hat{\rho}_1$ может оказаться смещенной оценкой ρ_1; иначе говоря, оно может не стремиться к ρ_1 при увеличении n. Таким образом, следует избегать сбора функционально зависимых наблюдений, чтобы выборочные средние были несмещенными оценками. Например, если бы $a(t)$ выбиралась для каждой миллионной операции, выполненной центральным процессором, центральный процессор всегда оказывался бы занятым, и мы получили бы ошибочный результат $\rho_1 \approx \hat{\rho}_1 = 1$.

Заметим, что для получения функционально независимых наблюдений необязательно их производить через случайные интервалы времени. Можно использовать и периодическую выборку, если приняты меры, чтобы частота выборки не совпала с целым кратным частоты гармоники, присутствующей в измеряемых данных. Например, если центральный процессор периодически прерывается по сигналам часов реального времени, считывая их состояние с частотой равной или кратной частоте часов, может наступить синхронизация наблюдений с часами. Так как центральный процессор непосредственно после приема прерывания всегда оказывается занятым, это может привести к ошибочному заключению $\rho_1 \approx \hat{\rho}_1 = 1$.

Какое минимальное количество наблюдений нужно произвести, чтобы сделать приемлемой оценку величины ρ_1 с помощью $\hat{\rho}_1$? В вероятностных терминах этот вопрос эквивалентен вопросу, каково минимальное значение n, удовлетворяющее неравенству

$$\text{Prob}\,(|\hat{\rho}_1 - \rho_1| < e) \geqslant K, \tag{2.3}$$

где K называется *доверительным уровнем*, а e — *доверительным интервалом* для $\hat{\rho}_1$. Заметим, что размер доверительного интервала в действительности равен $2e$.

По центральной предельной теореме мы имеем

$$N\left(\frac{e\sqrt{n}}{\sigma_1}\right) - N\left(\frac{-e\sqrt{n}}{\sigma_1}\right) \geqslant K, \tag{2.4}$$

где $N(x)$ означает значение стандартной нормальной функции распределения в точке x.

Пусть z выбрано так, чтобы для случайной переменной со стандартным нормальным распределением вероятность находиться в $(-z, z)$ была равна K. Тогда получим

$$N(z) = \frac{1+K}{2}, \quad N(-z) = \frac{1-K}{2}, \tag{2.5}$$

и из (2.4)

$$n \geqslant \left(\frac{\sigma_1 z}{e}\right)^2. \tag{2.6}$$

Соотношение (2.6) показывает, что количество наблюдений пропорционально квадрату доверительного интервала. Так как, вообще говоря, значение σ_1 не известно, его можно заменить *выборочным стандартным отклонением*

$$\hat{\sigma}_1 = \sqrt{\frac{\sum_{i=1}^{n} (a_i - \hat{\rho}_1)^2}{n-1}}. \tag{2.7}$$

Случайная переменная $(\hat{\rho}_1 - \rho_1)/(\hat{\sigma}_1/\sqrt{n})$ подчиняется t-распределению Стьюдента с $n-1$ степенями свободы. Если все $h+1$ переменных $x, y_1, \ldots, y_h$ независимы друг от друга и подчиняются стандартному нормальному распределению, то случайная переменная

$$t_h = \frac{x}{\sqrt{(1/h)\sum_{i=1}^{h} y_i^2}} \tag{2.8}$$

подчиняется t-распределению Стьюдента с h степенями свободы. t-распределение симметрично относительно среднего, равного нулю, и имеет дисперсию, равную $h/(h-2)$ при $h>2$. Заметим, что t-распределения, соответствующие разным числам степеней свободы, отличаются друг от друга, хотя и имеют несколько общих свойств. При увеличении степеней свободы t-распределения становятся все ближе к нормальному. Распределение величины $(\hat{\rho}_1 - \rho_1)/(\hat{\sigma}_1/\sqrt{n})$ имеет $n-1$ степень свободы, так как из n разностей $a_i - \hat{\rho}_1$ в (2.7) только $n-1$ независимы. Это обусловлено тем, что, согласно (2.2), они должны удовлетворять условию $\sum_{i=1}^{n} (a_i - \hat{\rho}_1) = 0$.

Применяя (2.6) с σ_1, приближенным величиной $\hat{\sigma}_1$, правильнее использовать вместо z величину z', которая подчинена t-распределению с $n-1$ степенями свободы (тогда как z подчинена стандартному нормальному распределению). Таким образом, z' определяется соотношениями $t_{n-1}(z') = (1+K)/2$ и $t_{n-1}(-z') = (1-K)/2$.

Значение величины z, определенной в (2.5), можно найти по таблицам функции стандартного нормального распределения, приведенным во многих статистических справочниках. Аналогично, значения z', определенные выше, можно найти в таблицах t-распределения.

Пример 2.6. Состояние центрального процессора периодически считывается для установления ρ_1. Пусть (неизвестное) поведение $a(t)$ будет таким, как изображено на рис. 2.12. Чтобы выборка была периодической, необходимо производить наблюдения в моменты, делящие интервал $(0, t_0)$ на равные части. Таким образом, если производятся три измерения, это будут моменты $t=25$, 50, 75, так как $t_0=100$; если $n=4$, моменты измерения будут $t=20$, 40, 60, 80. Таблица 2.11 содержит значения $\hat{\rho}_1$, которые получились бы для различных n. Заметим, что истинное значение коэффициента использования центрального процессора, которое легко получить непосредственно из рис. 2.12, равно 0.76. Истинная дисперсия распределения a, которую можно вычислить как средний квадрат отклонения от среднего ρ_1, равна $\sigma_1^2 = 0.182$. Стандартное отклонение равно $\sigma_1 = 0.427$.

Таблица 2.11

Результаты экспериментов с периодической выборкой, выполненных на состояниях центрального процессора, изображенных на рис. 2.12 (см. пример 2.6)

Число наблюдений n	Выборочное среднее $\hat{\rho}_1$	Выборочное стандартное отклонение $\hat{\sigma}_1$	Доверительный интервал для $K=0.95$, $e=\hat{\sigma}_1 z'/\sqrt{n}$
3	1.0	0.0	±0.0
4	1.0	0.0	±0.0
5	0.8	0.447	±0.55
6	0.666	0.516	±0.54
7	0.857	0.377	±0.35
8	0.75	0.462	±0.38
9	0.888	0.333	±0.25
15	0.733	0.457	±0.25
31	0.741	0.444	±0.16
50	0.76	0.431	±0.12

В табл. 2.11 также приведены выборочные стандартные отклонения, вычисленные по формуле (2.7) для различных значений n. Примем доверительный уровень $K=0.95$. Иначе говоря, мы хотим найти такую ширину доверительного интервала для $\hat{\rho}_1$, что вероятность для ρ_1 попасть вне этого интервала с центром $\hat{\rho}_1$ равна 5%. Это вычисление можно проделать, взяв знак равенства в (2.6) и решая полученное уравнение относительно e. Так как в практических случаях известно только $\hat{\sigma}_1$, следует заменить z на z' в (2.6). Столбец e в табл. 2.11 показывает эти доверительные интервалы для различных значений n. Например, если проведено 50 измерений, значение ρ_1 попадает в интервал 0.76 ± 0.12, т. е. в (0.64, 0.88) с вероятностью 0.95.

Можно вычислить из (2.6) минимальное количество наблюдений для уменьшения доверительного интервала до ±0.02, т. е. $\hat{\rho}_1-0.02\leqslant\rho_1\leqslant\hat{\rho}_1+0.02$ с вероятностью 0.95. Используя лучшую оценку σ_1 в табл. 2.11 ($\hat{\sigma}_1=0.431$) и значение z (поскольку z' стремится к z при росте n), получим $n\geqslant1784$. Если необходимо $e=\pm0.01$, то $n\geqslant7136$. Если $K=0.99$ и $e=\pm0.01$, то $n\geqslant12324$.

Если нужно измерять коэффициент использования центрального процессора и другие величины в течение периода времени фиксированной длительности, минимальная частота выборки определяется минимальным значением n из (2.6). Если нет ограничений на длительность сеанса измерения, частота выборки может быть взята произвольно, при условии что она не совпадает с какой-либо гармоникой периодического компонента измеряемой величины. Действительно, для данного доверительного уровня точность оценок зависит только от числа наблюдений n. Однако читатель должен был бы заметить, что это верно, только если распределения наблюдаемых величин не зависят от времени — иначе говоря, если последовательность наблюдений можно считать стационарной. Стохастическая последовательность X_i ($i=1, 2, \ldots$) называется *стационарной*, если совместные функции распределения $F_{X_jX_{j+1}\cdots X_{j+h}}(x_j, x_{j+1}, \ldots, x_{j+h})=\mathrm{Prob}\,(X_j\leqslant x_j, X_{j+1}\leqslant x_{j+1}, \ldots, X_{j+h}\leqslant x_{j+h})$ равны при всех $j=1, 2, \ldots$ и для всех $h=1, 2, \ldots$. Это эквивалентно требованию, чтобы все совместные функции распределения были инвариантны относительно сдвига по времени.

Очень часто полагают, что последовательности наблюдений параметров рабочей нагрузки можно моделировать стационарными временными рядами. В вычислительных центрах или вычислительных обслуживающих организациях это может оказаться не так. Рассмотрим очень простой параметр рабочей нагрузки — число заданий, поступающих в единицу времени на установку пакетной обработки. Если мы построим график интенсивности поступления заданий как функции времени, кривая, вероятно, будет демонстрировать относительно высокочастотные флуктуации на интервалах, скажем, порядка 1 часа. На многих установках удается также обнаружить периодические и квазипериодические вариации с большими периодами (день, неделя, месяц, квартал). Наконец, вероятно, удастся обнаружить еще более медленный апериодический дрейф, обычно в сторону больших интенсивностей поступления заданий. На протяжении интервалов времени, или много меньших или много больших по отношению к упомянутым вариациям, рабочая нагрузка (а следовательно, и система), вероятно, далека от стационарной. Ситуация может оказаться ближе к стационарной на интервалах промежуточной длительности, выбранной с учетом периодических вариаций. Например, мы можем ожидать этого, если рассмотрим один и тот же одночасовой интервал в течение нескольких последовательных дней недели или один и тот же день недели для нескольких последовательных недель. Однако всегда следует пытаться проверить справедливость предположения о стационарности поведения системы экспериментальными средствами.

Выборочные методы были введены в области исследования программного обеспечения Коленсом в 1968 г. [см. Коленс (1971)]. Одним из выборочных измерителей, выпускаемым корпорацией «Бул и Бэббидж», является «Программа оценки, использующая конфигурацию» (CUE).

Эта программа оценки состоит из двух частей: считывателя и анализатора. Во время сеанса измерения считыватель периодически вызывается с помощью системного интервального таймера и собирает данные из таблиц операционной системы, к которым он имеет доступ. Максимальная частота выборки равна частоте интервального таймера. Пользователь может выбрать дробные от нее частоты для уменьшения искажений. Собранные данные считыватель накапливает на внешнем запоминающем устройстве для последующего сжатия анализатором. Эти данные включают состояния центрального процессора, процессоров и устройств ввода-вывода, имена дисковых устройств, выполняющих операции поиска, имена нерезидентных модулей операционной системы, находящихся в памяти в момент считывания состояния системы.

Таблица 2.12

Пример выходной информации программы CUE

(a) *Сообщение об использовании оборудования*

Испытанное оборудование	Продолжительность, с	Процент общего времени
ЦП занят	2679.84	37.22
ЦП занят в режиме супервизора	303.84	4.22
Канал занят	3060.72	57.46
ЦП занят и канал занят	1514.16	21.03
ЦП занят, каналы свободны	1355.94	18.82
ЦП ждет, канал занят	2622.96	36.43
ЦП ждет, занят только канал 1	1779.84	24.71
Заняты канал 1 и канал 2	181.44	2.52
.	.	.
.	.	.
.	.	.

(b) *Сообщение о конкурирующих запросах на ввод-вывод*

Число запросов на ввод-вывод	Процент времени	Накопленный процент времени
0	3.51	3.51
1—6	7.32	10.83
7—12	19.87	30.70
13—18	54.92	85.62
19—24	7.43	93.05
25—30	6.95	100.00

(c) *Сообщение о движениях головок*
Устройство 131

Число цилиндров	Число перемещений с пересечением этого числа цилиндров	Время перемещения головки, мс	Процент общего времени
1	845	21125	2.08
.	.	.	.
.	.	.	.
.	.	.	.
60	700	50400	4.96
.	.	.	.
.	.	.	.
.	.	.	.

Продолжение

Число цилиндров	Число перемещений с пересечением этого числа цилиндров	Время перемещения головки, мс	Процент общего времени
141	1944	204120	20.09
142	579	61374	6.04
.	.	.	.
.	.	.	.
.	.	.	.

Адреса цилиндров	Число перемещений между этими двумя адресами	Время перемещения головки, мс	Процент общего времени
87 178	1119	117495	11.56
124 184	700	50400	4.96
36 177	527	55335	5.45
41 183	446	47276	4.65
124 125	338	8450	0.83
38 179	298	31290	3.08
. .	.	.	.
. .	.	.	.
. .	.	.	.

(d) *Сообщение об обращениях к супервизору*

Идентификатор обращения	Количество загрузок	Процент от всех загрузок	Накопленный процент
IGC0007B	61	20.40	20.40
IGG01910	41	13.71	34.11
IGG0191B	19	6.35	40.46
.	.	.	.
.	.	.	.
.	.	.	.
IGC0003D	1	0.33	100.00

Источник: Холтвик (1971).

Таблица 2.12 демонстрирует отрывок отчета, подготовленного анализатором. Данные в таблице были получены за два сеанса измерения длительностью по 1 часу. Времена этих двух сеансов были выбраны так, чтобы они попадали в ежедневный интервал пика нагрузки. Данные собирались с интервалом 33.33 мс. Так как суммарное время измерения составляло 2 часа, то для каждой измеряемой величины было проделано 216 000 измерений. Часть (a) отчета показывает значения

тех же коэффициентов использования и перекрытия, измерения которых с помощью измерителя с фиксированной программой было описано в примере 2.2 (см. табл. 2.3).

Сравнение данных, полученных программой CUE, с данными, полученными во время сеанса работы аппаратного измерителя [см. Питерсон (1974)], показывает, что на исследованной установке CUE дает точные результаты почти для всех каналов ввода-вывода. Однако значения ρ_1, полученные программой CUE, были в среднем на 7% ниже зарегистрированных аппаратным измерителем, значения ρ_{1p} (символику см. в табл. 2.3) были в среднем на 14% выше, а значения ρ_{1s} приблизительно на 21% ниже. Эти ошибки можно объяснить, если принять во внимание тот факт, что, когда центральный процессор занят и находится в состоянии супервизора, прерывания блокируются. Таким образом, при этом CUE лишается возможности получать данные, и вероятность того, что CUE обнаружит центральный процессор в состоянии задачи (или свободным), выше, чем в случае, если бы прерывания не блокировались. Это пример ошибки, обусловленной неслучайными, функционально зависимыми наблюдениями.

Относительные частоты состояний системы, характеризующихся различными количествами запросов на ввод-вывод (как уже обслуженных, так и ожидающих обслуживания) приведены в части (b) отчета. Для каждого дискового устройства с подвижными головками часть (c) отчета дает гистограмму числа цилиндров, пройденных в течение поиска, и список числа операций поиска между различными парами цилиндров. Наконец, в части (d) отчета содержится количество операций загрузки в память каждого нерезидентного модуля операционной системы.

Другим выборочным измерителем корпорации «Бул и Бэббидж» является «Система оценки проблемных программ» (PPE). Этот измеритель будет обсужден вкратце в разд. 9.3.1.

2.5. ИЗМЕРИТЕЛЬНЫЕ СИСТЕМЫ

Обычно измерители разрабатываются для существующих систем, для которых они затем и применяются. В результате такого подхода разработчики и пользователи измерителей часто сталкиваются с серьезными проблемами из-за ограничений, налагаемых на измерители структурой и реализацией измеряемой системы. Например, интересующие нас события могут оказаться практически недоступными, искажения могут быть недопустимо велики или убедиться в достоверности собранных данных может оказаться невозможным. С другой стороны, модификация системы с целью сделать ее более удобной для измерений — задача трудная, дорогая и довольно рискованная.

Большая часть этих проблем исчезла бы или хотя бы потеряла остроту, если бы об измерителях позаботились на этапе конструирования системы, а еще лучше, если бы система и ее измерительные средства разрабатывались вместе. Последний подход позволил бы также использовать измерители на завершающих этапах создания системы (см. гл. 8). Несколько исследовательских проектов разрабатывали эту проблему и накопили некоторый опыт. Измерительные средства систем Multics [см. Зальцер и Джинтелл (1970)] и PRIME [см. Феррари (1973)] уже были упомянуты в предыдущих разделах. В этих разработках одной из главных задач было прогнозирование (в момент, когда система еще не полностью специфицирована) событий и величин, которые предстоит наблюдать в будущем. Решения, принятые в этих двух проектах, могут быть описаны в виде различных комбинаций специализированных (фиксированных) измерителей для данной системы (особенно в системе PRIME) и измерителей общего типа. Фиксированные измерители встраивались для регистрации событий, будущая важность которых была ясна с самого начала. С другой стороны, более гибкие измерители должны были принять на себя заботу об остальных событиях, в частности о тех, которые должны обнаруживаться в работах по оценке, требующих временных измерений.

Когда измерители создаются вместе с системой, требования измерительных средств могут оказывать некоторое влияние на конструкцию системы. Например, к набору команд машины могут быть добавлены специальные команды, подобные приведенным на рис. 2.9. В систему могут быть встроены специальные счетчики, регистры, шины и команды доступа к памяти. Структура операционной системы и даже монтажная схема должны разрабатываться до некоторой степени с учетом интересов измерения.

Когда измерительные средства добавляются к существующей системе, возможно влияние только в одном направлении. Это означает, что архитектура системы влияет на измерители, часто накладывая ограничения на их возможности, но не наоборот.

В некоторых случаях ограничения на конструкцию измерительных средств являются следствием системных механизмов защиты. Цели измерения и защиты, как мы видели в разд. 2.3.4, противоположны. Экспериментаторы часто предпочли бы получать информацию, которая по своей природе не может быть доступна без нарушения секретности информации пользователей или ослабления защиты системы, которая должна обеспечивать техническую сторону требований секретности. Измерительные средства системы PRIME, машины, для которой одной из главных целей было обеспечение высокой секретности между процессами, являются примером ограничений, обусловленных правилами защиты, которые было решено не изменять для целей измерения. Например, в системе PRIME никакой процесс не может управлять другим, так как их области полностью изолированы.

Забота о средствах измерений на этапе разработки облегчает измерения на этапах испытания, настройки и работы системы, ведет к лучшему пониманию системы и делает ее более удобной для наших нужд. Более того, эта забота может стать необходимой, если существующая тенденция встраивать все более замысловатые управляющие механизмы в операционные системы сохранится. Проведение измере-

ний индексов производительности и параметров рабочей нагрузки в реальном времени и введение этих индексов в контрольные механизмы операционной системы важны для изготовления замкнутой системы управления с обратной связью. Подходящей аналогией, уже использованной для измерителей с запоминаемой программой, является аналогия с управлением промышленным процессом. Если объектом управления является процесс переработки информации, то модуль операционной системы выступает в роли управляющего устройства. Этот модуль пытается оптимизировать выполнение системы, изменяя параметры системы (в частности стратегию управления ресурсами) и иногда некоторые свойства рабочей нагрузки. Например, в условиях перегрузки можно уменьшить число пользователей диалоговой системы, запретив вход новых пользователей или автоматически прекратив диалог с некоторыми из ранее работавших пользователей. Выбор величин, которые требуется измерить и модифицировать в замкнутой системе с обратной связью обсуждается с общей точки зрения в разд. 7.4.2.

Все операционные системы содержат некоторые фиксированные программные измерители, на показаниях которых основываются решения о размещении ресурсов системы. Например, стратегия планирования многих систем пакетной обработки требует, чтобы задания выполнялись в соответствии с приоритетами и максимальными запросами ресурсов (время центрального процессора, пространство в памяти, число магнитных лент), указанными программистом в карте управления заданием. Однако системы, включающие алгоритм динамического управления с обратной связью, нуждаются в значительно более обширной измерительной системе, обеспечивающей непрерывное наблюдение. Проблемы стабильности, возникающие при таком подходе, изучены пока еще недостаточно. Методы и результаты теории автоматического управления могут оказаться применимыми к проектированию операционных систем, если удастся сформулировать подходящие модели системы и рабочей нагрузки.

2.6. ПЛАНИРОВАНИЕ ИЗМЕРИТЕЛЬНЫХ ЭКСПЕРИМЕНТОВ

2.6.1. Постановка задачи

В разд. 2.1 мы показали, что одним из основных шагов этапа планирования эмпирической работы по оценке является планирование эксперимента. *Эксперимент* состоит из набора эмпирических (или имитационных) тестов, выполняемых для получения ответов на вопросы, возникающие в ходе работы по оценке. Работа может потребовать многих экспериментов, а эксперимент в свою очередь состоит из ряда *сеансов*, или *прогонов*. Термин *сеанс* чаще применяется для измерений, а *прогон* чаще используется для имитационного моделирования. В течение сеанса или прогона накапливается информация о поведении системы и, возможно, о рабочей нагрузке. Поскольку рабочая нагрузка флуктуирует, количество наблюдений, которое требуется получить

для каждой интересующей нас величины, должно быть таково, чтобы распределения для этих величин и их моменты могли быть оценены с достаточной точностью. Таким образом, продолжительность сеанса будет зависеть от необходимого числа наблюдений. Проблема определения длительности сеанса и времени его выполнения будет исследована в разд. 2.6.4.

Индексы производительности системы зависят от числа параметров установки. Принимая терминологию, используемую в статистическом планировании эксперимента, назовем *факторами* параметры, заметно влияющие на значение данного индекса производительности. Некоторые из факторов относятся к *количественным*, например запрос времени центрального процессора для заданий, составляющих определенную рабочую нагрузку. Однако следует также принимать во внимание и *неколичественные* факторы, такие, как стратегия планирования центрального процессора. Поэтому «значения» фактора будут называться *уровнями* — термин, применимый и к неколичественным факторам.

Сеанс обычно определяется конкретной комбинацией уровней факторов. Если имеется три фактора A, B и C, уровни которых обозначены $A1, A2, \ldots, B1, B2, \ldots, C1, C2, \ldots$, то тройка ($Ai$, Bj, Ck) определяет сеанс, во время которого уровень A равен Ai, уровень B равен Bj, а уровень C равен Ck.

Эксперимент длительностью в один сеанс достаточен для работы по оценке, если нужно рассмотреть только одну конфигурацию системы и один тип рабочей нагрузки. Это может иметь место, например, если измерения проводятся с целью выяснить, обеспечивает ли данная система при данной рабочей нагрузке удовлетворительную производительность, т. е. отвечает ли она определенным минимальным требованиям.

Эксперименты длительностью в несколько сеансов необходимы, если предстоит определить влияние определенных факторов на производительность системы или производится оптимизация системы последовательными итерациями. Основными проблемами, возникающими при планировании этих экспериментов, являются

a) идентификация факторов,
b) выбор их уровней для каждого сеанса.

Эти проблемы будут обсуждаться в разд. 2.6.2 и 2.6.3 соответственно.

Никакой эксперимент не достоверен, если он не воспроизводим. В области измерения производительности вычислительных машин очень трудно спланировать воспроизводимый эксперимент. Эта проблема будет обсуждаться в гл. 5, так как воспроизводимость экспериментов на вычислительных системах определенно зависит от выбора и контроля рабочей нагрузки. Заметим, что из флуктуаций рабочей нагрузки во времени не обязательно должно следовать непостоянство уровней факторов, характеризующих рабочую нагрузку. Например,

если параметры рабочей нагрузки описываются как стационарные случайные переменные, их распределения, не зависящие от времени, можно рассматривать как уровни соответствующих факторов. Таким образом, различные сеансы можно было бы определить как отличающиеся по крайней мере одним из этих распределений.

2.6.2. Идентификация факторов

Первым шагом в планировании эксперимента из нескольких сеансов является идентификация факторов. Это, вообще говоря, трудная задача. Часто придание количественной формы влиянию определенных факторов на индекс производительности само по себе представляет проблему, требующую работы по оценке для своего решения. Как мы увидим в разд. 2.7, обнаружение в результате эксперимента того, что некоторый индекс не зависит заметно от определенного параметра, приводит нас к заключению, что этот параметр не является фактором. Однако, когда эксперимент еще планируется, мы часто должны выбирать параметры, которые, вероятно, будут важными факторами, еще не зная их истинного веса.

В этой задаче идентификации исследователи должны полагаться на опыт, знание системы и на свою интуицию. Число параметров, которые, как мы полагаем, будут влиять на индекс производительности, может оказаться очень большим. С другой стороны, как мы увидим в разд. 2.6.3, стоимость эксперимента довольно резко возрастает с увеличением числа рассматриваемых факторов. Таким образом, приходится проводить отбор, и он должен быть проделан тщательно. К сожалению, этот отбор осложняется внутренней взаимозависимостью параметров, о которой часто известно мало или ее наличие только предполагается.

Полезно классифицировать факторы, отобранные для эксперимента, на *первичные* и *вторичные*. Первичные факторы — это те параметры, в изучении влияния которых на производительность системы экспериментатор заинтересован непосредственно. Вторичные факторы принимаются во внимание только постольку, поскольку их влиянием, по-видимому, нельзя пренебречь. Однако количественная оценка этого влияния не входит в цели исследования; скорее, наоборот, экспериментатор часто пытается уничтожить или ослабить это влияние посредством тщательного планирования эксперимента, чтобы эффекты первичных факторов можно было ясно видеть и интерпретировать.

Введем два дополнительных полезных определения: *управляемых* и *наблюдаемых* факторов. Управляемый фактор — это такой фактор, уровни которого выбираются экспериментатором. Наблюдаемый фактор — это фактор, уровни которого можно измерить в ходе эксперимента. Если система во время сеанса измерения управляется своей обычной *производственной рабочей нагрузкой*, параметры рабочей нагрузки весьма слабо поддаются управлению. В этом случае факторы, характеризующие рабочую нагрузку, можно только наблюдать. Эти факторы

могли бы быть управляемыми, если бы пользователи были полностью исключены и система была полностью ориентирована на эксперимент и управлялась *искусственной рабочей нагрузкой*. Мы увидим в гл. 3, что все факторы в имитационных экспериментах могут управляться и наблюдаться, даже если степень управляемости рабочей нагрузки зависит от типа модели, использованной для представления рабочей нагрузки.

Пример 2.7. Обсудим теперь идентификацию факторов при планировании эксперимента над памятью со страничной организацией, следуя Цао и др. (1972). *Страничная организация* — это один из методов реализации виртуальной памяти (см. разд. 1.2). В своем простейшем варианте страничное пространство виртуальных адресов есть упорядоченное множество n адресов или ячеек информации, индексируемых последовательными натуральными числами от 0 до n—1. Эти номера суть виртуальные адреса элементов данных. Это множество делится на подмножества фиксированного размера, содержащие последовательные виртуальные адреса и называемые *виртуальными страницами*. Когда программа загружается в это виртуальное пространство, ее набор слов делится на подмножества, соответствующие виртуальным страницам. Все слова, содержащиеся в виртуальной странице, составляют *страницу программы*. *Страничное устройство* — это устройство внешней памяти, на котором может разместиться вся программа во время выполнения. Обмен информацией между устройством внешней памяти и основной памятью производится страницами. Основная память делится на *действительные страницы*, имеющие тот же размер, что и их виртуальные двойники. Каждая страница, будучи загружена в память, занимает одну действительную страницу.

Как мы уже видели в разд. 1.2, в системе с виртуальной памятью в основную память обычно загружена лишь часть программы. Когда программа ссылается на фрагмент информации, не находящийся в памяти, мы получаем отказ. В системе со страничной организацией он называется *страничный отказ*, так как вызывает перенос в основную память целой страницы, содержащей слово, на которое была ссылка. Если для поступающей страницы нет места, одна из страниц, находящаяся в памяти, должна быть вытолкнута. Выбор такой страницы производится операционной системой в соответствии с *алгоритмом замещения*.

Рассмотрим большую программу, скажем компилятор, которая одна выполняется на машине со страничной организацией памяти, причем основная память машины меньше программы. Время прохождения программы сильно зависит от числа страничных отказов, возникающих во время ее выполнения, так как время, необходимое для переноса страницы с вращающегося страничного устройства, скажем с барабана, в основную память, приблизительно на четыре десятичных порядка больше, чем время доступа к основной памяти. Таким образом, число страничных отказов является важным индексом производительности для программы. Мы хотим спланировать эксперимент с целью определить, какие параметры системы и программы оказывают наиболее сильное влияние на количество страничных отказов. Как нам идентифицировать параметры, среди которых следует искать факторы? Как выбрать сами факторы?

Количество страничных отказов, возникающих во время выполнения программы, зависит от поведения самой программы и от стратегии системы в управлении иерархией памяти. В однопрограммном случае, который мы рассматриваем, размер пространства, отведенного в памяти данной программе, постоянен и равен доступной части памяти. Стратегия управления этим пространством может рассматриваться как комбинация стратегий *загрузки* и *замещения*. Существуют два главных типа стратегии загрузки: *загрузка по требованию* и *опережающая загрузка*. При загрузке по требованию страница загружается в память только тогда, когда появляется ссылка на одно из имеющихся в ней слов. Опережающая загрузка, наоборот, ориентирована на предсказание будущих потребностей выполняемой программы и может загружать страницы прежде, чем к ним возникнут обращения. Используются также смешанные стратегии, например отведенная область памяти может быть в начале выполнения заполнена

(типичная опережающая загрузка), а затем может загружаться по требованию. Стратегии замещения были вкратце введены выше.

Итак, мы можем предполагать, что число страничных отказов зависит от размера памяти, стратегии загрузки и стратегии замещения.

Параметрами, которые могут воздействовать на поведение данной программы, являются входные данные и порядок, в котором ее части (например, подпрограммы) размещены в пространстве виртуальных адресов. Влияние последнего параметра обусловлено тем, что различные упорядочивания, вообще говоря, обусловливают различное содержимое страниц. Таким образом, при возникновении страничного отказа в память загружаются различные части программы, и содержимое памяти в момент одного и того же обращения при двух таких упорядочиваниях оказывается разным.

Среди параметров мы должны выделить факторы эксперимента. Существуют различные критерии их выбора. Некоторые из параметров могут быть отброшены из-за того, что они явно не имеют отношения к нашему исследованию или из-за того, что их влияние на индекс производительности хорошо известно. Экспериментаторы могут также исключить параметры, которыми они не интересуются и которые полностью контролируются. Например, мы будем предполагать, что нас не интересует влияние стратегии загрузки на количество страничных отказов. Таким образом, эту стратегию нужно будет выбрать и сохранять постоянной на протяжении эксперимента. Мы выберем загрузку по требованию.

Четыре фактора, выбранные для эксперимента над памятью со страничной организацией, перечислены в табл. 2.13. Заметим, что все они являются первичными и контролируемыми факторами. ■

2.6.3. Выбор уровней для каждого сеанса

Идентифицировав контролируемые факторы, следует выбрать их уровни. Уровни некоторого фактора должны соответствующим образом заполнять диапазон изменения этого фактора. С другой стороны, общее количество уровней следует минимизировать из экономических соображений. Таким образом, обычно приходится идти на компромисс между этими двумя противоположными требованиями. Иногда количество рассматриваемых уровней для некоторого фактора бывает ограничено целями эксперимента. Например, если целью является сравнение результатов применения двух разных стратегий, соответствующий фактор будет, естественно, иметь два уровня.

Как только заданы уровни каждого фактора, следует выбрать комбинации этих уровней, имеющие смысл для эксперимента. С использованием символики, введенной в разд. 2.6.1, эта задача может быть поставлена как выбор для n-факторного эксперимента наборов по n чисел (Ai, Bj, Ck, . . .) уровней, соответствующих каждому сеансу. Если l_A, l_B, l_C, . . .— числа уровней для факторов A, B, C, . . . соответственно, то мы имеем $1 \leqslant i \leqslant l_A$, $1 \leqslant j \leqslant l_B$, $1 \leqslant k \leqslant l_C$. Так как каждому сеансу измерения соответствует набор из n чисел, стоимость эксперимента будет линейно зависеть от количества наборов. *Факторный план*, включающий все различные комбинации уровней, часто влечет за собой недопустимое количество сеансов, и его следует избегать, насколько это возможно. Это количество равно произведению $l_A l_B l_C$ . . .; если все n факторов имеют одинаковое число уровней l, произведение равно l^n. Например, если каждый из четырех факторов примера 2.7 имеет три уровня, требуется $3^4 = 81$ сеанс для факторного эксперимента.

Таблица 2.13

Факторы и уровни, выбранные для эксперимента над памятью со страничной организацией

Программа: компилятор с ФОРТРАНа
Стратегия загрузки: по требованию

Факторы		Уровни	
Название	Символ	Имя	Описание
Размер памяти	M	LARGE	24 страницы
		MEDIUM	20 страниц
		SMALL	16 страниц
Алгоритм замещения	R	LRU	Удаляется наиболее давно не используемая страница
		FIFO	Первым пришел, первым ушел
		RAND	Случайный
Входные данные (задача компилируемой программы)	I	SMALL	55 операторов, 3.88 с (время ЦП)
		MEDIUM	215 операторов, 18.05 с (время ЦП)
		LARGE	595 операторов, 149.55 с (время ЦП)
Порядок подпрограмм в виртуальной памяти	S	GROUP	Основан на частоте обращений подпрограмм друг к другу
		FREQY	Основан на интенсивности замены страниц
		ALPHA	Алфавитный порядок

Источник: Цао и др. (1972).

К сожалению, как мы увидим в разд. 2.7, факторные эксперименты являются единственными экспериментами, позволяющими определить эффекты взаимодействия между факторами. Таким образом, если предстоит анализировать такие взаимодействия, то количество факторов и уровней, включенных в эксперимент, должно быть выбрано с еще большей тщательностью, чем в других случаях. Действительно, влияние этих количеств на стоимость эксперимента намного больше, чем если бы эффекты взаимодействия вовсе не изучались. Когда взаимодействия считают отсутствующими или их эффектами пренебрегают вследствие малости по сравнению с эффектами отдельных факторов, для уменьшения стоимости эксперимента можно выбрать *частичный*, или неполный, *факторный план*.

Если комбинация уровней для каждого сеанса выбирается случайно, мы имеем *полностью рандомизированный план*. Использование этого типа плана подразумевает, что все параметры, не включенные в эксперимент, поддерживаются постоянными. Если такое предположение сделать нельзя, как это часто бывает при изучении вычислительных систем, параметры, не остающиеся постоянными на протяжении эксперимента, должны быть включены в план как вторичные факторы. Их вариации часто поддаются до некоторой степени управлению посредством тщательного выбора комбинаций уровней управляемых факторов и условий проведения каждого сеанса. Цель этого выбора, как

будет показано более подробно в разд. 2.7, состоит в том, чтобы отделить воздействия на индексы производительности от следующих *источников вариаций*: первичных факторов, вторичных факторов, ошибок эксперимента.

Предположим, что имеется только один вторичный фактор, например время дня, в которое наблюдается система, обрабатывающая свою нормальную рабочую нагрузку. Тогда можно группировать наблюдения в *блоки*, характеризующиеся большей однородностью экспериментальных условий, чем весь эксперимент. Например, если предстоит изучать воздействие некоторых алгоритмов планирования на среднее время ответа в диалоговой системе и наблюдения накапливают в течение нескольких дней, блоки могут содержать наблюдения, сделанные в одни и те же интервалы суток каждого дня. Внутри каждого блока уровни первичных факторов упорядочены, вообще говоря, в случайном порядке, чтобы уменьшить возможные систематические изменения от сеанса к сеансу. Такая организация называется *рандомизированным блочным планом*. Пример такого плана приведен в табл. 2.14 для только что приведенного эксперимента.

Таблица 2.14

Рандомизированный блочный план

День	Время дня		
	10 час утра	1 час дня	4 час дня
1	FCFS	RR	SJF
2	RR	FCFS	LJF
3	SJF	LJF	RR

FCFS: первым пришел — первым обслужен
RR: циклическое повторение
SJF: самое короткое задание — первым
LJF: самое длинное задание — первым

Если имеется два вторичных фактора, можно организовать блоки в соответствии с такими их комбинациями, которые создают более однородные экспериментальные условия для каждого сеанса. *Латинский план* для эксперимента с одним первичным фактором A, имеющим l_A уровней, получается при выборе l_A уровней для каждого из двух вторичных факторов B и C при расположении l_A^2 блоков в виде квадратной матрицы и приписывании блокам уровней фактора A так, чтобы каждый уровень A появлялся только один раз в каждой строке и каждом столбце. Такое приписывание не однозначно, и существует некоторая свобода в его выборе. Латинский план для эксперимента, описанного в предыдущем абзаце, иллюстрируется табл. 2.15. Заметим, что этот эксперимент состоит из 16 сеансов, тогда как соответствующий факторный эксперимент потребовал бы $4^3=64$ сеанса.

Противоположностью факторному эксперименту является *эксперимент с изменением факторов по одному*. Этот подход, естественный для исследователя, состоит в изменении факторов по одному, тогда как остальные $n-1$ факторов поддерживаются постоянными. Таким образом, эксперимент разбивается на n отдельных экспериментов, каждый из которых имеет целью изучение эффектов одного фактора. Для факторов A, B, C, . . . общее число сеансов будет $l_A+l_B+l_C+\ldots$. Например, эксперимент (см. табл. 2.15), в котором 3 фактора имеют по 4 уровня каждый, потребует 12 сеансов, если будет спланирован

Таблица 2.15

Латинский план

День недели	Время дня			
	9 утра	11 утра	1 час дня	3 час дня
Понедельник	FCFS	RR	SJF	LJF
Среда	RR	SJF	LJF	FCFS
Пятница	SJF	LJF	FCFS	RR
Суббота	LJF	FCFS	RR	SJF

согласно подходу «изменение факторов по одному». Заметим, что этот метод для большинства практических целей дает неполную информацию. Дело в том, что вообще невозможно предсказать эффект изменения фактора A, когда остальные факторы фиксируются на уровнях, отличающихся от выбранных для эксперимента, в котором менялся только фактор A. Следовательно, подход «изменение факторов по одному» можно использовать только тогда, когда известно, что нет взаимодействия между факторами — другими словами, когда эффект каждого фактора не зависит от остальных. Иначе выводы из эксперимента могут быть ошибочными.

Как было отмечено Гренандером и Цао (1972), нефакторный эксперимент, по-видимому, более устраивает исследователей, чья цель — сравнить различные системы, алгоритмы или стратегии. С другой стороны, те планы и задачи оптимизации, в которых анализ возможных взаимодействий весьма желателен или обязателен, требуют проведения факторного эксперимента.

Пример 2.8. В примере 2.7 были идентифицированы параметры и отобраны факторы для эксперимента со страничной организацией памяти (см. табл. 2.13). Программа, на которой будет проведен эксперимент,— компилятор с языка ФОРТРАН. Отбор уровней факторов следует делать с учетом типа выбранной программы, области пространства параметров, которую мы хотим охватить нашими экспериментами, а также с учетом неизбежных физических и экономических ограничений.

Чтобы исключить большую часть источников изменчивости, внешних по отношению к программе, сеансы измерения должны производиться на системе, специально отведенной для этой цели. Это решение, конечно, делает эксперимент намного более

точным, но и намного более дорогим. Другим аспектом, определяющим стоимость эксперимента, является наше желание анализировать взаимодействие между факторами, так как оно требует факторного плана. Если взять два уровня на фактор, то это даст $2^4 = 16$ сеансов. Однако два уровня не представили бы адекватно область исследования. Три уровня на фактор потребуют 81 сеанс, а четыре уровня 256 сеансов. Допустим, что наш бюджет еще позволяет провести 81 сеанс, и выберем для каждого фактора по три уровня так, что они будут соответствовать *малому*, *среднему* и *большому* количеству страниц. Эти уровни перечислены в табл. 2.13 в порядке предполагаемого возрастания количества страничных отказов.

Заметим, что M — количественный фактор, тогда как остальные три — качественные. Выбор уровней такого фактора, как I (входные данные), обычно очень трудно обосновать. В этом случае игнорируются все характеристики компилируемой программы, кроме ее длины. Таким образом, в качестве трех уровней I должны быть выбраны короткая, средняя и длинная программы из рабочей нагрузки установки (среди программ на ФОРТРАНе), и тогда I становится в некотором смысле количественным фактором.

В отношении уровней фактора S предполагаем, что GROUP будет давать наименьшее количество страничных отказов, так как этот уровень получается группировкой программ, вызывающих друг друга наиболее часто. Второй уровень, FREQY, соответствует упорядочению подпрограмм по убыванию частоты страничных отказов при предыдущих выполнениях компилятора. Третий уровень, ALPHA, это версия компилятора, в которой подпрограммы загружаются в память в алфавитном порядке.

Читателю следует заметить, что выбор уровней оказывает сильное влияние на стоимость эксперимента. В факторном плане каждый уровень фактора используется в сочетании со всеми уровнями всех остальных факторов; в данном примере каждый уровень используется в 27 сеансах. Таким образом, уровень, вызывающий слишком большое число страничных отказов, будет оказывать нежелательное воздействие на треть сеансов в этом эксперименте.

Заметим также, что природа нашего индекса производительности такова, что достаточно только одного сеанса на комбинацию уровней. Если измеритель, подсчитывающий страничные отказы, работает правильно, никаких ошибок эксперимента возникнуть не должно. Однако, если бы индекс был статистическим дескриптором (таким, как среднее распределения), нам, возможно, пришлось бы повторять каждый сеанс несколько раз, чтобы уменьшить ошибки эксперимента. Это дополнительное затруднение следует, разумеется, принимать во внимание при выборе факторов и уровней. ■

2.6.4. Время и длительность сеанса

Из-за непостоянства рабочей нагрузки каждое отдельное измерение индекса производительности или параметра установки очень часто оказывается бесполезным. В общем случае целью сеанса измерения является оценка интеграла по времени (например, числа событий) или оценка распределений интересующих величин и их моментов.

Как долго должен длиться сеанс? Когда следует проводить сеанс?

Второй вопрос уместен, только если систему наблюдают в процессе обработки производственной рабочей нагрузки. В этом случае ответ зависит от решаемой проблемы, от характеристик использования установки, а также от ответа на первый вопрос. Например, очевидно, что, если должна изучаться производительность системы при большой нагрузке, следует проводить измерения тогда, когда, по-видимому, достигаются пики интенсивности использования системы. Однако если каждый сеанс может длиться целый день, то проблема выбора времени начала измерения заменяется на проблему выбора дней для сеансов.

Как было показано в разд. 2.6.3 (табл. 2.14 и 2.15), время сеанса в некоторых экспериментах может диктоваться выбранным планом.

На вопрос о продолжительности сеанса вообще гораздо труднее ответить. В ряде экспериментов эта продолжительность является одним из индексов производительности, который нужно измерять, или прямо связана с индексом. Так оказывается, например, если искусственная рабочая нагрузка, состоящая из N заданий, используется для управления системой, особенно если средняя пропускная способность системы находится среди индексов производительности, подлежащих определению (см. пример 1.1). В этих случаях проблема выбора подходящей продолжительности сеанса не возникает, но встает аналогичный, настолько же важный вопрос: насколько велико должно быть значение N и как должна быть составлена искусственная рабочая нагрузка? Этот вопрос подробно обсуждается в гл. 5.

В этом разделе мы ограничим наше внимание экспериментами, в которых исследователь прямо управляет продолжительностью сеансов измерения. Данные, собранные в этих экспериментах, можно рассматривать как временные ряды, состоящие из наблюдений определенных величин. Ряд наблюдений величины u может рассматриваться как выборка из стохастической последовательности. Если эта последовательность стационарна, ее среднее $\bar{u}$ не зависит от времени. Оценкой $\bar{u}$ является среднее $\bar{u}_n$ по временному ряду $u_1, \ldots, u_n$. Если точность этой оценки возрастает с ростом n, говорят, что последовательность *эргодическая*. Будем предполагать, что последовательности, с которыми мы будем иметь дело, эргодические. Таким образом, если заданы максимальная допустимая ошибка оценки (доверительный интервал) и минимальная вероятность того, что истинное среднее $\bar{u}$ лежит внутри этого интервала вблизи выборочного среднего $\bar{u}_n$ (доверительный уровень оценки), то существует минимальный размер исследуемой выборки (см. разд. 2.4.3). Этот размер соответствует минимальной длительности сеанса.

Следует заметить, что существуют исследовательские работы, даже рассматриваемого здесь типа, для которых определение минимальной продолжительности сеанса не важно. Например, если эксперимент состоит из одного или нескольких сеансов, выбрать продолжительность эксперимента настолько большой, чтобы заведомо удовлетворить требованиям точности, может оказаться дешевле, чем оценить минимальную приемлемую продолжительность сеанса. В других случаях (в частности, когда ожидаемое число сеансов велико или когда адекватность рабочей нагрузки, использованной для управления системой, и справедливость выводов из эксперимента должны быть гарантированы) необходимо или по крайней мере весьма желательно вычислять минимальную приемлемую длительность сеанса.

Как показано в разд. 2.4.3, если наблюдения, сделанные в течение сеанса измерения, не коррелированы, дисперсия выборочного среднего около истинного среднего будет σ^2/n, где σ^2 есть истинная дисперсия распределения, а n — размер выборки. Таким образом, минимальная

длительность сеанса может быть получена из неравенства (2.6). Если учитывать только знак равенства, (2.6) выражает минимальное значение n, назовем его $n_{\min}$ как функцию от σ^2, доверительного интервала e и доверительного уровня. Даже при этих предположениях длительность сеанса очень зависит от размера доверительного интервала, так как $n_{\min}$ обратно пропорционально квадрату этой величины. Например, если разделить доверительный интервал пополам, длительность должна увеличиться в четыре раза.

Однако в основном из-за нехватки неделимых ресурсов, из-за которых конкурируют задания, наблюдения в вычислительных системах часто оказываются автокоррелированными. Например, время ожидания в очереди для задания сильно коррелировано с временами ожидания для заданий с высоким приоритетом в той же самой очереди. Следовательно, дисперсия выборочного среднего обычно отличается от σ^2/n. Если, однако, распределение $\bar{u}_n$ является нормальным, неравенство (2.4) можно применять при условии, что σ_1^2/n будет заменено на $\mathrm{Var}(\bar{u}_n)$. Таким образом, $n_{\min}$ будет минимальным значением n, удовлетворяющим неравенству (аналог (2.6))

$$\mathrm{Var}(\bar{u}_n) \leqslant \left(\frac{e}{z}\right)^2, \tag{2.9}$$

где e есть доверительный интервал, а z соотносится с доверительным уровнем, как показано в (2.5).

К сожалению, распределение $\bar{u}_n$ обычно неизвестно. Если мы можем сделать единственное предположение, что $\bar{u}_n$ имеет конечное среднее и конечную дисперсию, то можно записать неравенство Чебышева [см. например, Феллер (1968)] в виде

$$\mathrm{Prob}(|\bar{u}_n - \bar{u}| < e) \geqslant 1 - \frac{\mathrm{Var}(\bar{u}_n)}{e^2} \quad (e > 0). \tag{2.10}$$

Это неравенство вместе с (2.3) дает условие

$$\mathrm{Var}(\bar{u}_n) \leqslant e^2(1-K), \tag{2.11}$$

где K есть доверительный уровень для $\bar{u}_n$. Значение $n_{\min}$, полученное из (2.11), обычно много больше, чем полученное из (2.9) для тех же значений e и K. Если сделать дополнительные предположения относительно распределения $\bar{u}_n$, величину $n_{\min}$ можно уменьшить. Если эти предположения слабее предположения о нормальности, результирующее значение $n_{\min}$ заключено между величинами, получающимися из (2.9) и (2.11).

Как оценить $\mathrm{Var}(\bar{u}_n)$? Вспомнив определение дисперсии $\mathrm{Var}(x) = E[(x-\bar{x})^2]$ и определение ковариации двух случайных переменных $\mathrm{Cov}(x, y) = E[(x-\bar{x})(y-\bar{y})]$, где $E(w)$ означает среднее (или математическое ожидание) величины w, читатель может легко проверить, что если $u_1, \ldots, u_n$ образуют ряд наблюдений, извлеченный из стационарной стохастической последовательности со средним $\bar{u}$, а $\bar{u}_n$ является

средним из n наблюдений, дисперсия $\bar{u}_n$ и $\bar{u}$ равна

$$\mathrm{Var}(\bar{u}_n) = \frac{1}{n^2}\left[\sum_{i=1}^{n}\mathrm{Var}(u_i) + \sum_{\substack{i,\,j=1\\ i\neq j}}^{n}\mathrm{Cov}(u_i, u_j)\right]. \tag{2.12}$$

Если $u_1, \ldots, u_n$ независимы и одинаково распределены со средним $\bar{u}$ и дисперсией σ^2, то (2.12) принимает вид

$$\mathrm{Var}(\bar{u}_n) = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}, \tag{2.13}$$

так как $\mathrm{Cov}(u_i, u_j) = 0$ для всех i, j при $i \neq j$. Однако, если наблюдения коррелированы, (2.13) применять нельзя, как и было сказано выше. Существуют различные методы *оценки дисперсии*, помимо очевидного, использующего (2.12). Мы опишем коротко методы автоковариаций, независимых копий и подвыборок.

Метод *автоковариаций* основан на уравнении, которое можно получить прямо из (2.12). Нам придется сделать предположение, что $u_1, \ldots, u_n$ распределены одинаково со средним $\bar{u}$ и дисперсией σ^2. Кроме того, в ряде наблюдений, который мы рассматриваем, ковариация между любыми двумя наблюдениями должна быть конечной и зависеть только от расстояния τ между ними по времени:

$$R_\tau = \mathrm{Cov}(u_i, u_{i+\tau}) = \mathrm{Cov}(u_i, u_{i-\tau}) \tag{2.14}$$

для всех i ($1 \leqslant i \leqslant n$) и всех $\tau = 1, \ldots, n-1$. Если удовлетворены эти предположения, то формулу (2.12), содержащую все автоковариации, можно переписать в виде

$$\mathrm{Var}(\bar{u}_n) = \frac{\sigma^2}{n} + \frac{2}{n}\sum_{\tau=1}^{n-1}\left(1 - \frac{\tau}{n}\right)R_\tau. \tag{2.15}$$

Так как автоковариации R_τ уменьшаются с ростом τ, количество m автоковариаций, подлежащих оценке, часто удается сделать значительно меньше, чем n. Оценка величин R_τ в реальном времени ($\tau = 1, \ldots, m-1$) может накапливаться на протяжении сеанса измерения посредством запоминания только отдельных наблюдений, собранных на предыдущем интервале длиной m наблюдений. Следовательно, $\mathrm{Var}(\bar{u}_n)$ можно оценивать динамически и сеанс может быть прекращен, как только ее значение удовлетворит выбранному условию (например, (2.11)). Так как программный измеритель, вероятно, вызовет недопустимые искажения, этот подход, по-видимому, реально подходит лишь для аппаратного измерителя с запоминаемой программой.

Другой метод оценки $\mathrm{Var}(\bar{u}_n)$ — метод *независимых копий*. Применяя его, мы пытаемся воссоздать условия, при которых справедливо (2.13). Каждый сеанс измерений заменяется h независимыми подсеансами, во время каждого из которых собирают q наблюдений $u_{j1}, \ldots, u_{jq}$ ($j = 1, \ldots, h$). Таким образом, общее количество наблюдений оказывается равным $n = hq$. В силу предположения о независимости h

средних

$$\bar{u}_{q(j)} = \frac{1}{q} \sum_{i=1}^{q} u_{ji} \quad (j = 1, \ldots, h) \tag{2.16}$$

являются независимыми случайными переменными, и, следовательно, по центральной предельной теореме, их распределение около $\bar{u}$ стремится к нормальному с ростом h. Кроме того, выборочное среднее, определенное в (2.16),

$$E(\bar{u}_{q(j)}) = \frac{1}{h} \sum_{j=1}^{h} \bar{u}_{q(j)} = \bar{u}_n \tag{2.17}$$

стремится к $\bar{u}$ при росте q. Таким образом, для достаточно больших h и q, мы имеем из (2.13)

$$\mathrm{Var}(\bar{u}_n) = \frac{1}{h} \mathrm{Var}(\bar{u}_{q(j)}). \tag{2.18}$$

Значения $\mathrm{Var}(\bar{u}_{q(j)})$ могут быть оценены с помощью выборочной дисперсии, как в (2.7):

$$\hat{\mathrm{Var}}(\bar{u}_{q(j)}) = \frac{1}{h-1} \sum_{j=1}^{h} (\bar{u}_{q(j)} - \bar{u}_n)^2. \tag{2.19}$$

Метод *подвыборок* очень похож на метод независимых копий. Он состоит в разделении длинного, непрерывного сеанса на h подсеансов и в сборе q наблюдений в каждом из них. Средние по подвыборкам не независимы, но если q достаточно велико, можно считать их ковариации малыми. Если эти предположения (которые следует проверить, оценив ковариации по собранным данным) справедливы, то можно использовать (2.18) для оценки $\mathrm{Var}(\bar{u}_n)$.

Оценка доверительных интервалов в реальном времени возможна в данном и в предыдущем методах. Удобной стратегией выбора q, применимой для обоих методов, является следующая: накапливать среднее и дисперсии для подвыборок с малыми q; затем постепенно увеличивать q, рассматривая подсеансы удвоенного, утроенного, учетверенного размера, до тех пор, пока не будет достигнута желаемая точность. При этом необходимо помнить, что значение q в методе независимых копий должно быть большим, чтобы сделать $\bar{u}_n$ несмещенной оценкой $\bar{u}$. В методе подвыборок q также должно быть велико, чтобы сделать ковариации между средними последовательных подсеансов достаточно малыми.

Чтобы вычислить доверительный интервал в реальном времени, не внося слишком больших искажений, следует использовать аппаратный измеритель с запоминаемой программой. С таким оборудованием можно оканчивать сеанс, как только размер интервала станет приемлемо малым. Читателю нужно, однако, иметь в виду, что описанные вы-

ше методы оценки дисперсии следует применять в измерительных экспериментах с осторожностью. Одна из трудностей состоит в том, что оценка доверительного интервала в реальном времени может оказаться ненадежной в присутствии нестационарной рабочей нагрузки. Большая часть проблем, возникающих при измерении систем, управляемых производственной рабочей нагрузкой, оказывается определенно менее острой в имитационных экспериментах (см. гл. 3), которыми легче управлять, или в измерительных экспериментах, управляемых искусственной рабочей нагрузкой. Действительно, приведенные выше методы оценки доверительного интервала и других величин гораздо шире используются в моделировании, чем в измерении.

Пример 2.9. Необходимо оценить производительность четырех алгоритмов планирования для диалоговой системы и сравнить их друг с другом. Эти алгоритмы перечислены в табл. 2.14. Первичным индексом производительности будет среднее время ответа системы. Так как исследователей интересует в основном, как обрабатываются задачи с большим количеством взаимодействий различными алгоритмами планирования при различных нагрузках, то подготавливается сценарий, состоящий из последовательности команд редактирования. Этот сценарий должен быть введен в систему внешним драйвером. Этот драйвер является устройством, которое имитирует отдельного пользователя, воспроизводя его время обдумывания и его скорость ввода сообщений через терминал. Кроме того, это устройство способно измерять время ответа на команды сценария и накапливать эти величины для дальнейшей обработки.

Чтобы до некоторой степени отделить эффекты производственной рабочей нагрузки от эффектов алгоритмов планирования, мы введем два вторичных фактора: «время дня» и «день недели», и используем латинский план (табл. 2.15). Какова должна быть продолжительность этих 16 сеансов? Иными словами, сколько наблюдений величин времени ответа должно быть произведено в каждом сеансе?

Существенно отметить, что эти вопросы неявно подразумевают стационарность измеряемого стохастического процесса. В противном случае выражение среднего для распределения времени ответа в виде одного числа и оценка его доверительного интервала окажутся бессмысленными. С другой стороны, об этом аспекте процессов, характеризующих поведение вычислительной системы, известно очень мало. Чтобы определить, в какой степени справедливо предположение о стационарности процесса, необходимо проверить для системы выполнение обширных статистических тестов. В нашем обсуждении мы будем, как принято обычно, исходить из этого предположения.

То, что эксперимент состоит из 16 сеансов, оправдывает нашу заботу об уменьшении продолжительности сеанса до минимума, который еще обеспечивает желательную точность оценки индекса производительности. Природа рассматриваемого примера такова, что достаточен доверительный интервал $\pm 5\%$ для среднего времени ответа при доверительном уровне 95%. Другими словами, мы не считаем существенной разницу в 10% между производительностями двух алгоритмов планирования.

Для определения продолжительности каждого сеанса используем метод подвыборок. Сделаем предположение, что продолжительность, оказавшаяся достаточной для одного из сеансов, подойдет (возможно, умноженная на коэффициент запаса, больший единицы) для всех сеансов. Это, разумеется, довольно сильное предположение, так как различные сеансы происходят с рабочей нагрузкой, обладающей различными характеристиками. Альтернативой было бы принять это предположение только для сеансов, происходящих в одно и то же время дня. В этом случае с каждым столбцом табл. 2.15 будет связана независимо определенная продолжительность. Другой, лучшей альтернативой была бы оценка дисперсии в реальном времени в ходе каждого сеанса и окончание сеанса по достижении желательной точности.

Итак, один из сеансов выбирается для целей оценки дисперсии, например сеанс с применением алгоритма FCFS (первым пришел — первым обслужен) в 9 часов в понедельник. Используемый сценарий состоит из 15 команд редактирования. Драйвер

может быть запрограммирован таким образом, что сценарий будет вводиться повторно без перерывов, так что первая команда будет следовать за пятнадцатой, как если бы она была шестнадцатой. Каждый сценарий занимает в среднем 2.5 минуты реального времени. Таким образом, за 2-часовой сеанс драйвер, вероятно, сможет ввести 48 последовательных копий сценария.

Таблица 2.16

Средние времена ответа в секундах для каждой копии сценария, пары и четверки копий в эксперименте примера 2.9

Одиночный сценарий	Пара	Четверка	Одиночный сценарий	Пара	Четверка
1.21			2.27		
1.67	1.44		1.93	2.1	
1.71		1.53	2.19		2.1075
1.53	1.62		2.04	2.115	
2.03			1.92		
2.15	2.09		1.97	1.945	
1.88		2.02	1.65		1.8125
2.02	1.95		1.71	1.68	
1.75			1.89		
1.84	1.795		1.70	1.795	
1.61		1.6375	1.62		1.6725
1.35	1.48		1.48	1.55	
1.43			1.55		
1.64	1.535		1.39	1.47	
1.52		1.5075	1.45		1.515
1.44	1.48		1.67	1.56	
1.17			1.62		
1.42	1.295		1.77	1.695	
1.64		1.5225	1.88		1.7725
1.86	1.75		1.82	1.85	
1.68			1.93		
1.91	1.795		2.09	2,01	
1.73		1.875	2.24		2,105
2.18	1.955		2.16	2.2	

Пусть 48 средних значений времени ответа за 15 команд в каждом из 48 экземпляров совпадают с перечисленными в хронологическом порядке в табл. 2.16. Общее среднее $\bar{t}_n$ равно 1.756 с. Чтобы убедиться, что эти 15 наблюдений, собранных для каждой копии сценария, можно рассматривать как подвыборку, вычислим коэффициент автокорреляции между средними $\bar{t}_j$ и $\bar{t}_{j+1}$ ($j=1, \ldots, 47$) последовательных копий.

Этот коэффициент определяется как отношение $\mathrm{Cov}(\bar{t}_j, \bar{t}_{j+1})/\mathrm{Var}(\bar{t}_j)$. Заметим, что для краткости мы заменили $\bar{t}_{15(j)}$ более коротким обозначением $\bar{t}_j$. Оценка $\mathrm{Var}(\bar{t}_j)$, полученная из (2.19), составляет приблизительно 0.074. Величину $\mathrm{Cov}(\bar{t}_j, \bar{t}_{j+1})$ можно оценить, применив ее определение. Так как эта оценка равна 0.048, результирующий коэффициент автокорреляции приблизительно равен 0.65.

Коэффициент автокорреляции по определению заключен между -1 и $+1$. Если бы средние величины для последовательных копий были независимы, этот коэффициент был бы равен 0. Коэффициент 0.65 указывает на довольно сильную автокорреляцию между последовательными средними. Чтобы уменьшить ее, следует выделять подвыборки большего размера. Если мы повторим приведенные выше вычисления для средних по подвыборкам, составленным из пар последовательных копий этого же сценария, то получим коэффициент автокорреляции 0.56. Это еще слишком много.

Продолжим группировку последовательных копий. Если каждая подвыборка содержит четыре копии, коэффициент автокорреляции для 12 таких подвыборок составит 0.08. В контексте наших ограниченных требований к точности эта степень автокорреляции может считаться пренебрежимо малой. Таким образом, наши подвыборки будут состоять из четырех последовательных копий сценария, или, что то же, из последовательности 60 команд.

Сколько подвыборок должен содержать сеанс? Из (2.18) и (2.9) получаем неравенство

$$h \geqslant \mathrm{Var}(\bar{t}_j)\left(\frac{z}{e}\right)^2, \tag{2.20}$$

аналогичное (2.6). Величина $\mathrm{Var}(\bar{t}_j)$, соответствующая оценке, основанной на 12 четверках (см. табл. 2.16), приблизительно равна 0.052, e равняется $0.05\,\bar{t}_n = 0.087$, а $z = 2.201$ для $K = 95\%$ и $n = 12$. Следовательно, (2.20) дает оценку $h \geqslant 33.7$. Если мы выберем $h = 36$, соответствующая продолжительность сеанса должна быть около 6 часов. Даже посвящая целых 2 часа с 9 до 11 утра каждый понедельник сбору подвыборок FCFS, мы должны затратить 3 недели на сбор 36 подвыборок, а весь эксперимент займет около 3 месяцев. Чтобы уменьшить это время, следует воспользоваться методами уменьшения дисперсии (см. разд. 3.4.2). Кроме того (или вместо того) мы могли бы перепланировать эксперимент иначе — так, чтобы использовать еще и вторники и четверги, а также вечера, чтобы группировать их вместе с субботами.

Заметим, что, если бы общая длительность сеанса была ограничена 2 часами (т. е. 12 подвыборками), доверительный интервал для среднего времени ответа, соответствующий доверительному уровню 95%, был бы около $\pm 8.3\%$. Для доверительного уровня 90% интервал был бы $\pm 6.7\%$. ■

2.7. ИНТЕРПРЕТАЦИЯ РЕЗУЛЬТАТОВ ИЗМЕРЕНИЙ

2.7.1. Сжатие данных и методы описания

При обсуждении измерителей мы уже видели, что, если результаты измерений не предъявляются или не используются в реальном времени, может оказаться удобным накапливать необработанные данные, а обрабатывать их позже по окончании сеанса. Это рекомендуется особенно в тех случаях, когда предварительная обработка вызвала бы излишние помехи во времени. Итак, если цели исследования не полностью определены в самом начале или если нельзя пренебречь вероятностью возникновения новых, непредвиденных вопросов в ходе исследования или в результате него, то необработанные данные обычно могут дать больше ответов на различные вопросы, чем данные, обработанные при сборе с существенной потерей информации.

В любом случае данные, собранные в течение сеанса, почти всегда требуют некоторого преобразования, прежде чем они будут предъявлены экспериментатору. Измерители обычно снабжаются программами сжатия данных и генерации отчетов, которые подсчитывают статистику индексов производительности и других параметров установки и представляют их в табличной и графической форме.

Часто эти средства сжатия и представления результатов предлагают экспериментатору на выбор несколько выходных форматов, величин и статистических показателей. Удобной и гибкой организации собранных данных можно достичь, накапливая их в базе данных и предоста-

вив пользователям язык запросов, посредством которого они смогут запрашивать базу данных и взаимодействовать с ней. Пример того, насколько эффективной может быть такая организация, можно найти у Купермана и др. (1972). Созданный ими пакет программ сбора статистики (SGP) включает язык запросов (подобный PL/1), позволяющий пользователям определять создаваемые подмножества базы данных, а также получать сводки и более подробную статистику относительно пользователя, программы и других атрибутов задания и системы.

Интересно отметить операционную разницу между этим подходом и диалоговым измерительным подходом, представленным информатором и системой SMT (см. разд. 2.4.2). При первом подходе данные собирают в необработанном виде, поэтому они настолько подробны, что позволяют получать ответы на большое количество вопросов. С другой стороны, второй подход позволяет экспериментатору быстро и безопасно начать сеанс измерений, как только возникнет нужда в этом, и измерять именно те величины, которые интересуют его в данное время.

Форма представления результатов может сильно облегчить (или затруднить) их интерпретацию. Одна из важных проблем в данной области связана обычно с большим количеством чисел, которые надо

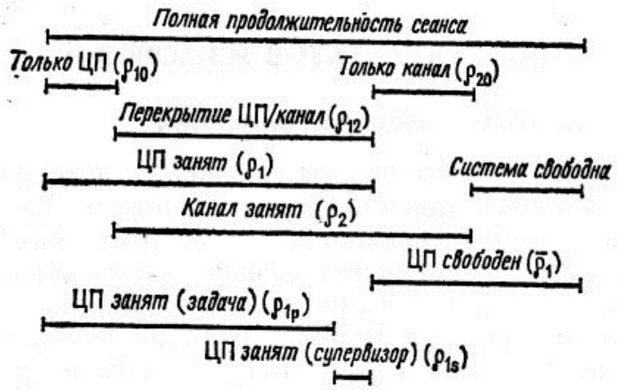


Рис. 2.13. Диаграмма Ганта (профиль использования системы), соответствующая данным табл. 2.3.

одновременно рассматривать при интерпретации результатов, что делает затруднительной работу с двумерными таблицами и диаграммами. Поскольку существование взаимосвязей, структур и трендов может оказаться незамеченным (пока результаты не упорядочены или не изображены подходящим образом) возможно, что будут обнаружены только те зависимости, существования которых ожидают. Среди типов графического изображения результатов, которые предположительно помогут экспериментатору в интерпретации результатов измерений, здесь будут кратко введены диаграммы Ганта и графики Кивиата. Их применение в работах по оценке описано в гл. 7.

Диаграммы Ганта можно использовать для совокупного представления характеристик использования компонентов системы. В этом случае их часто называют *профилями использования системы*. Простой профиль системы, состоящей из одного центрального процессора и одного канала ввода-вывода, данные измерения которого приведены в табл. 2.3, изображен на рис. 2.13. Читатель сам убедится в том, что гораздо легче понять временные взаимоотношения между рассматриваемыми величинами из профиля рис. 2.13, чем из чисел табл. 2.3. Это тем более верно, когда установка содержит несколько каналов и устройств. Профиль использования системы дает экспериментатору совокупное представление в графической форме состояний активности элементов установки и перекрытия их во времени. Этот метод представления специально ориентирован на мультипрограммные системы,

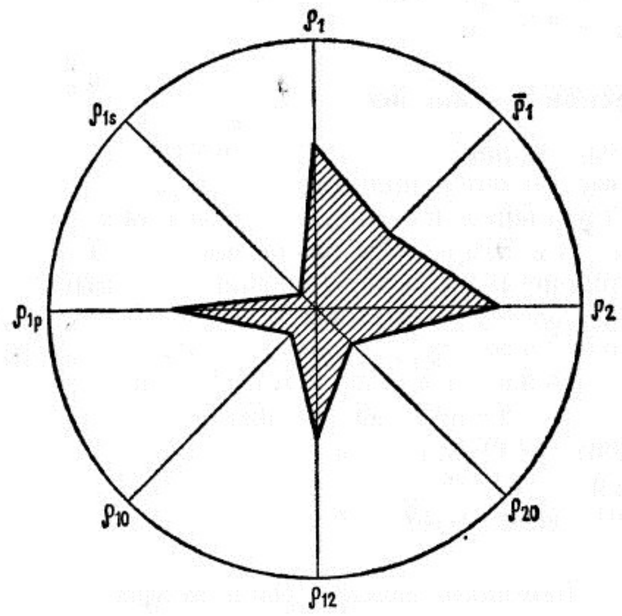


Рис. 2.14. График Кивиата, соответствующий данным табл. 2.3.

которые стараются использовать взаимную независимость различных компонентов системы с целью получения более высокой пропускной способности.

Данные табл. 2.3 можно изобразить с помощью кругового графика, радиусы которого используются как оси для представления изображаемых величин (см. рис. 2.14). Эта конспективная форма представления называется *графиком Кивиата*. Он использует способность экспериментатора распознавать образы и, следовательно, немедленно обнаруживать наличие некоторой проблемы или возможных причин ее возникновения, что открыть иным способом было бы трудно. Выбор величин, откладываемых по осям, вполне произволен. Распространенный подход, которому следует также рис. 2.14, состоит в выборе рав-

ного количества «хороших» и «плохих» величин и в изображении их на соседних осях. Чем больше «хорошая» величина и чем меньше «плохая», тем лучше. Таким образом, график Кивиата для хорошей системы будет похож на звезду, и наш глаз легко заметит разницу между некоторым изображением и правильной звездой.

Вообще, очень мало можно сказать об интерпретации результатов. Это в значительной степени мастерство, в котором можно достичь совершенства только практикой и изучая опыт других. Возможности успешной интерпретации определяются как интуицией, так и глубоким знанием структуры и организации системы. Несколько примеров интерпретации в частных случаях будет дано в гл. 6, 7 и 8. В двух последующих разделах будут введены (посредством примеров их применения к задачам оценки производительности) некоторые методы, пригодные для интерпретации результатов статистически спланированных экспериментов.

2.7.2. Дисперсионный анализ

Термин *дисперсионный анализ* обозначает группу методов анализа данных, которые можно использовать для установления относительного влияния различных источников изменения на значения индекса производительности. Эти методы разлагают общую дисперсию данного индекса на компоненты, соответствующие рассматриваемым источникам изменения. Если все факторы в эксперименте количественные, можно применить метод *регрессионного анализа*. Пример регрессионного анализа обсуждается в разд. 2.7.3. В данном разделе мы опишем применение методов дисперсионного анализа в случае, когда не все факторы количественные.

Пример 2.10. Рассмотрим эксперимент над системой со страничной организацией памяти, описанной в примерах 2.7 и 2.8, предполагая, что на протяжении экспери-

Таблица 2.17

Результаты двухфакторного эксперимента

		S			Средние по строкам ($\overline{F}_{i.}$)	Главные эффекты M (M_i)
		GROUP	FREQY	ALPHA		
M	LARGE	1.51	1.72	1.77	1.666	−0.629
	MEDIUM	1.68	2.39	2.73	2.266	−0.029
	SMALL	2.73	3.00	3.13	2.953	+0.658
Средние по столбцам ($\overline{F}_{.j}$)		1.973	2.370	2.543	2.295	Общее среднее ($\overline{F}_{..}$)
Главные эффекты S (S_j)		−0.322	+0.075	+0.248		

Источник: Цао и др. (1972).

...алгоритм замещения R и входные данные I остаются постоянными, скажем на ...ответствующих уровнях LRU и SMALL (см. табл. 2.13). В табл. 2.17 показаны $3^2=$ ... результатов (каждый является десятичным логарифмом общего числа страничных ..., возникших в соответствующем сеансе). Был выбран логарифм, поскольку ..., что влияние факторов на число страничных отказов мультипликативно ... ниже).

Вообще, фактор M имеет l_M уровней, а фактор S имеет l_S уровней; в нашем случае $l_S=l_S=3$. Кроме того, F_{ij} ($i=1, \ldots, l_M$; $j=1, \ldots, l_S$) есть значение индекса производительности (т. е. логарифм числа страничных отказов) для сеанса, соответствующего уровню i фактора M и уровню j фактора S. *Средние по строкам* задаются выражением

$$\overline{F}_{i.}=\frac{\sum_{j=1}^{l_S} F_{ij}}{l_S}, \tag{2.21}$$

...средние по столбцам —

$$\overline{F}_{.j}=\frac{\sum_{i=1}^{l_M} F_{ij}}{l_M} \tag{2.22}$$

...среднее по всем наблюдениям, также называемое *общим средним*,—

$$\overline{F}_{..}=\frac{\sum_{i=1}^{l_M}\sum_{j=1}^{l_S} F_{ij}}{l_M l_S}. \tag{2.23}$$

Главный эффект строки i (т. е. фактора M на уровне i) определяется как

$$M_i=\overline{F}_{i.}-\overline{F}_{..} \tag{2.24}$$

...аналогично, столбца j (т. е. фактора S на уровне j) — как

$$S_j=\overline{F}_{.j}-\overline{F}_{..}. \tag{2.25}$$

Заметим, что общее среднее, определенное в (2.23), можно рассматривать как ...среднее от средних по строкам или как среднее от средних по столбцам. Суммы отклонений этих средних от общего среднего равны нулю, как можно убедиться по ...ле 2.17:

$$\sum_{i=1}^{l_M} M_i=\sum_{j=1}^{l_S} S_j=0. \tag{2.26}$$

Отклонение F_{ij} от общего среднего можно записать как

$$F_{ij}-\overline{F}_{..}=(\overline{F}_{i.}-\overline{F}_{..})+(\overline{F}_{.j}-\overline{F}_{..})+(F_{ij}-\overline{F}_{i.}-\overline{F}_{.j}+\overline{F}_{..}). \tag{2.27}$$

Тождество (2.27) выражает отклонение как сумму главных эффектов M_i и S_j ...факторов на их соответствующих уровнях i и j и остаточного члена. Этот член ...представляет комбинированные эффекты взаимодействия M и S и экспериментальных ...ошибок. Если в каждом сеансе сделано только одно измерение, как в нашем случае, ...этот член нельзя разложить на два компонента. Поэтому мы назовем его *эффектом ...взаимодействия* MS_{ij}, предполагая при этом, что вклад в него ошибки эксперимента ...равен нулю или очень мал. В нашем эксперименте это предположение, по-видимому, ...удовлетворено, так как индекс производительности подсчитывается в дубле. Вообще, ...эти два компонента можно разделить, только если каждый сеанс дает h значений ...индекса производительности, как будет показано ниже в этом разделе.

Итак, (2.27) можно переписать в виде

$$F_{ij}-\overline{F}_{..}=M_i+S_j+MS_{ij}, \tag{2.28}$$

а из (2.27), (2.21) и (2.22) очевидно, что

$$\sum_{i=1}^{l_M} MS_{ij}=\sum_{j=1}^{l_S} MS_{ij}=0. \tag{2.29}$$

Следует заметить, что *взаимодействие* не есть синоним «зависимости между факторами». Эффект взаимодействия, определенный в (2.27), может быть ненулевым, даже если факторы независимы. Предположим, например, что между индексом производительности F^* и этими же двумя факторами существует мультипликативное функциональное соотношение (оба фактора для простоты считаем количественными):

$$F^*_{ij}=H\cdot Mi\cdot Sj, \tag{2.30}$$

где M_i и S_j есть i-й и j-й (количественные) уровни факторов M и S. Легко видеть, что индекс производительности F_{ij} не может быть выражен в виде суммы только двух главных эффектов. Следовательно, наличие взаимодействия может быть обусловлено как зависимостью, так и отсутствием аддитивности (или обеими причинами). Отсутствие аддитивности в некоторых случаях удается устранить подходящими преобразованиями данных. Например, в случае мультипликативного соотношения, как в (2.30), можно взять логарифмы от обеих частей:

$$\log F^*_{ij}=\log H+\log Mi+\log Sj. \tag{2.31}$$

Если существует зависимость между факторами, никакое преобразование данных ее не может устранить.

Дисперсия наблюдений относительно общего среднего пропорциональна

$$\sum_{i=1}^{l_M}\sum_{j=1}^{l_S}(F_{ij}-\overline{F}_{..})^2. \tag{2.32}$$

Согласно тождеству (2.28), (2.32) можно записать в виде

$$\begin{aligned}\sum_{i=1}^{l_M}\sum_{j=1}^{l_S} M_i^2+\sum_{i=1}^{l_M}\sum_{j=1}^{l_S} S_j^2+\sum_{i=1}^{l_M}\sum_{j=1}^{l_S} MS_{ij}^2+2\sum_{i=1}^{l_M}\sum_{j=1}^{l_S} M_iS_j+\\ +2\sum_{i=1}^{l_M}\sum_{j=1}^{l_S} M_iMS_{ij}+2\sum_{i=1}^{l_M}\sum_{j=1}^{l_S} S_jMS_{ij}=\\ =l_S\sum_{i=1}^{l_M} M_i^2+l_M\sum_{j=1}^{l_S} S_j^2+\sum_{i=1}^{l_M}\sum_{j=1}^{l_S} MS_{ij}^2.\end{aligned} \tag{2.33}$$

Заметим, что упрощение в (2.33) возможно, так как три перекрестных произведения равны нулю [см. (2.26) и (2.29)]. Три ненулевых члена в (2.33) называются *суммами квадратов* соответственно для фактора M (обозначается SSM), для фактора S (обозначается SSS) и для взаимодействия (обозначается SSI). Общая сумма квадратов в (2.32), обозначаемая SST, делится, следовательно, на три компонента:

$$SST=SSM+SSS+SSI, \tag{2.34}$$

которые представляют относительную важность трех рассматриваемых источников изменения.

Отношения SSM/SST, SSS/SST и SSI/SST обычно интерпретируются как доли общей дисперсии, «объясняемые» фактором M, фактором S и их взаимодействием MS

соответственно. Эти доли можно представить графически в виде *круговой* диаграммы, как на рис. 2.15. Так как эти три рассмотренных источника изменения полностью объясняют общую дисперсию, уравнение (2.28) можно интерпретировать как эмпирическую модель F_{ij}:

$$F_{ij}=\overline{F}_{..}+M_i+S_j+MS_{ij}. \tag{2.35}$$

Значения параметров модели, полученные из результатов табл. 2.17, приведены в табл. 2.18. Значение индекса производительности, соответствующее данной комбинации уровней, можно получить из (2.35). Например, десятичный логарифм от числа

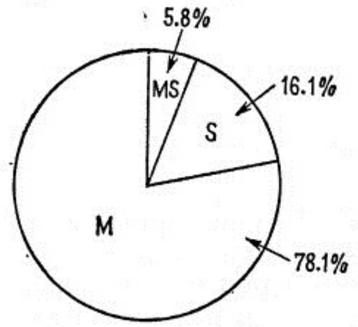


Рис. 2.15. Круговая диаграмма, демонстрирующая степень влияния размера памяти M, порядка подпрограмм S и их взаимодействия MS на изменения логарифма числа страничных отказов в эксперименте примера 2.10.

страничных отказов для памяти объемом 20 страниц и алфавитно упорядоченных подпрограмм равен $F_{23}=\overline{F}_{..}+M_2+S_3+MS_{23}=2.73$. Следовательно, мы имеем 536 страничных отказов.

Круговая диаграмма на рис. 2.15 показывает, что размер основной памяти, выделенной компилятору с ФОРТРАНа, несомненно, оказывает самое сильное влияние

Таблица 2.18

Параметры эмпирической модели F_{ij} в (2.35)

$\overline{F}_{..}=$ 2.295	$MS_{11}=$ 0.166
$M_1=-0.629$	$MS_{12}=-0.021$
$M_2=-0.029$	$MS_{13}=-0.144$
$M_3=$ 0.658	$MS_{21}=-0.264$
$S_1=-0.322$	$MS_{22}=$ 0.049
$S_2=$ 0.075	$MS_{23}=$ 0.216
$S_3=$ 0.248	$MS_{31}=$ 0.099
	$MS_{32}=-0.028$
	$MS_{33}=-0.071$

на количество замен страниц. Эффекты упорядочения подпрограмм существенно менее важны, но ими нельзя пренебречь. Взаимодействие MS объясняет около 6% общей дисперсии. Из-за его сравнительно малой величины мы можем сказать, что вклады этих двух факторов в логарифм числа страничных отказов почти аддитивны. Это эквивалентно утверждению, что их дисперсии влияют на число страничных отказов почти мультипликативно. Модель (2.35) представляет разложение логарифма индекса производительности на вклады источников изменения. Если мы будем игнорировать член, соответствующий взаимодействию, мы сможем выразить индекс производительности (который является антилогарифмом F_{ij}) в виде произведения антилогарифма остальных членов в (2.35). Таким образом, изменение уровня некоторого фактора будет воздействовать на число страничных отказов приблизительно пропорционально (это справедливо только для количественных факторов).

Если вкладом некоторого источника изменения пренебрегли на том основании, что его сумма квадратов очень мала по сравнению с SST, остальные источники больше

не могут полностью объяснить общую дисперсию. Например, Цао и Марголин (1972) в четырехфакторном эксперименте, описанном в примерах 2.7 и 2.8, смогли объяснить 98% общей дисперсии при рассмотрении только четырех основных эффектов и одного эффекта взаимодействия первого порядка (MS):

$$F_{ijhk}=\overline{F}_{....}+M_i+R_j+I_h+S_k+MS_{ik}+E_{ijhk}, \tag{2.36}$$

где E_{ijhk} — член, представляющий потерянные эффекты взаимодействия. Этот член также называется *остатком* для индекса $ijhk$ и является ошибкой оценки F_{ijhk} с помощью

$$\hat{F}_{ijhk}=\overline{F}_{....}+M_i+R_j+I_h+S_k+MS_{ik}. \tag{2.37}$$

Проверка остатков полезна для оценки степени соответствия эмпирической модели экспериментальным данным. Их значения могут указать на *выбросы*, т. е. на ошибочные значения. Кроме того, их знаки могут помочь обнаружить систематические ошибки, например потерю важного эффекта взаимодействия. ■

В конце примера 2.10 мы видели, что общая дисперсия не может быть полностью объяснена, если некоторые эффекты игнорируются. Это верно и при наличии ошибок эксперимента. Рассмотрим факторный эксперимент с двумя факторами A и B, и пусть индексом производительности будет величина, измеряемая с некоторой неопределенностью. Например, индексом могло бы быть среднее время ответа в диалоговой системе. Если мы копируем каждый сеанс h раз, как описано в разд. 2.6.4, то каждый элемент ij в таблице результатов ($i=1, \ldots, l_A$; $j=1, \ldots, l_B$) содержит h наблюдений $\overline{t}_{ijv}$ ($v=1, \ldots, h$).

Итак, мы строим эмпирическую модель

$$\overline{t}_{ijv}=\overline{t}_{...}+A_i+B_j+AB_{ij}+E_{ijv}, \tag{2.38}$$

и (2.34) принимает вид

$$SST=SSA+SSB+SSI+SSE. \tag{2.39}$$

Используя обозначения примера 2.10, можно представить суммы квадратов в правой части (2.39) в виде

$$\begin{aligned}
SSA &= hl_B\sum_{i=1}^{l_A}(\overline{t}_{i..}-\overline{t}_{...})^2,\\
SSB &= hl_A\sum_{j=1}^{l_B}(\overline{t}_{.j.}-\overline{t}_{...})^2,\\
SSI &= h\sum_{i=1}^{l_A}\sum_{j=1}^{l_B}(\overline{t}_{ij.}-\overline{t}_{i..}-\overline{t}_{.j.}+t_{...})^2,\\
SSE &= \sum_{i=1}^{l_A}\sum_{j=1}^{l_B}\sum_{v=1}^{h}(\overline{t}_{ijv}-\overline{t}_{ij.})^2.
\end{aligned} \tag{2.40}$$

Заметим, что отношение SSE/SST представляет часть общей дисперсии, объясняемую ошибками эксперимента.

Между эффектами A_i, B_j, AB_{ij}, появляющимися в эмпирической модели (2.38), сохраняются соотношения (2.26) и (2.29). Следователь-

но, только l_A-1 из l_A главных эффектов фактора A независимы: если известны l_A-1 значений A_i, величина оставшегося A_i определяется равенством (2.26). В таком случае говорят, что SSA имеет l_A-1 *степеней свободы*. Аналогично, число степеней свободы SSB равно l_B-1, и вследствие соотношений, аналогичных (2.29), число степеней свободы SSI равно $(l_A-1)(l_B-1)$. Например, легко проверить, что все MS_{ij} в табл. 2.18 можно вывести из любых четырех из них.

Из определения общего среднего в (2.23) и выражения для SST в (2.32) получим, что SST обладает hl_Al_B-1 степенями свободы. Уравнение (2.39) свидетельствует, что степени свободы аддитивны. Так, число степеней свободы SSE равно

$$hl_Al_B-1-(l_A-1)-(l_B-1)-(l_A-1)(l_B-1)=(h-1)\,l_Al_B. \tag{2.41}$$

Результат подтверждает утверждение, сделанное при обсуждении примера 2.10, что эффекты ошибок эксперимента нельзя отделить от эффектов взаимодействий, если есть только одно наблюдение на сеанс ($h=1$).

Отношение каждой суммы квадратов к соответствующему числу степеней свободы называется *средним квадратом*. Например, $MSA=SSA/(l_A-1)$. Средние квадраты полезны для проверки значимости результатов дисперсионного анализа. Из-за ошибок эксперимента влияние некоторого фактора или взаимодействия на индекс производительности, оказавшееся ненулевым, можем быть на самом деле несущественным (т. е. меньшим, чем влияние ошибок). В терминах статистического вывода, мы должны проверить сначала следующую гипотезу:

$$H_0'\colon AB_{ij}=0 \quad (i=1, \ldots, l_A;\ j=1, \ldots, l_B). \tag{2.42}$$

Для проверки гипотезы необходимо выбрать *уровень значимости* α. Уровень значимости есть вероятность того, что гипотеза отвергается, будучи верной. Ясно, что чем меньше α, тем выше доверие к результату проверки.

Если H_0' отвергается на данном уровне значимости, то велика вероятность существования взаимодействия между A и B. Следовательно, главные эффекты этих двух факторов также будут существенны. Если, наоборот, H_0' нельзя отвергнуть, то следует заключить, что A и B не взаимодействуют существенно и, возможно, что эффекты одного или обоих несущественны. Чтобы определить, верно ли это статистически, мы должны проверить две гипотезы

$$H_0''\colon A_i=0 \quad (i=1, \ldots, l_A) \quad \text{и} \quad H_0'''\colon B_j=0 \quad (j=1, \ldots, l_B). \tag{2.43}$$

Сделаем теперь следующие предположения:

a) $\overline{t}_{ijv}$ — случайные переменные с нормальным распределением;
b) E_{ijv} — взаимно независимые нормально распределенные случайные переменные со средним 0 и дисперсией σ_E^2 для всех i, j и v.

Если эти предположения удовлетворены и если гипотеза H_0' верна, можно показать, что отношение MSI/MSE — случайная переменная, подчиненная F-распределению с $(l_A-1)(l_B-1)$ и $(h-1)l_Al_B$ степенями свободы. Если имеется n_1 независимых случайных переменных x_i, подчиненных стандартному нормальному распределению, и n_2 случайных переменных y_j того же типа, то случайная переменная

$$F_{n_1, n_2} = \frac{(1/n_1)\sum_{i=1}^{n_1} x_i^2}{(1/n_2)\sum_{j=1}^{n_2} y_j^2} \tag{2.44}$$

подчиняется F-распределению с n_1 и n_2 степенями свободы. Большинство статистических справочников приводит таблицы F-распределения. Обычно элемент такой таблицы, соответствующий параметрам α, n_1, n_2 (мы обозначим его через $F_{\alpha;\, n_1,\, n_2}$) таков, что

$$\text{Prob}\,(F_{n_1,\, n_2} > F_{\alpha;\, n_1,\, n_2}) = \alpha. \tag{2.45}$$

Пусть H_0' верна. Тогда вероятность отвергнуть ее будет меньше или равна α. В силу (2.45) H_0' отвергается на уровне значимости α, если

$$\frac{MSI}{MSE} > F_{\alpha;\,(l_A-1)(l_B-1),\,(h-1)\,l_Al_B}. \tag{2.46}$$

Аналогично, H_0'' и H_0''' отвергаются на уровне значимости α, если

$$\frac{MSA}{MSE} > F_{\alpha;\,(l_A-1),\,(h-1)\,l_Al_B} \quad \text{и} \quad \frac{MSB}{MSE} > F_{\alpha;\,(l_B-1),\,(h-1)\,l_Al_B}. \tag{2.47}$$

Величины, позволяющие проверить значимость дисперсионного анализа, обычно организуют в таблицу типа табл. 2.19. Тест, описанный выше называется *F-тестом*. В разд. 2.7.3 F-тест применяется для определения значимости уравнения регрессии.

Предположения «a» и «b», приведенные выше, вероятно, часто не справедливы для экспериментов на вычислительных системах. Как было обнаружено в эксперименте, невыполнение предположения «a»

Таблица 2.19

Таблица дисперсионного анализа для двухфакторного эксперимента с h наблюдениями на каждую комбинацию уровней

Источник изменчивости	Сумма квадратов	Степени свободы	Средний квадрат	Вычисленное значение F
Фактор A	SSA	l_A-1	$MSA=SSA/(l_A-1)$	MSA/MSE
Фактор B	SSB	l_B-1	$MSB=SSB/(l_B-1)$	MSB/MSE
Взаимодействие AB	SSI	$(l_A-1)(l_B-1)$	$MSI=SSI/[(l_A-1)(l_B-1)]$	MSI/MSE
Ошибки E	SSE	$(h-1)\,l_Al_B$	$MSE=SSE/[(h-1)\,l_Al_B]$	
Общее T	SST	hl_Al_B-1		

не влияет серьезно на результат теста. Это верно также для части предположения «b» относительно равных дисперсий (если различия дисперсий не слишком велики). Во всяком случае, справедливость предположений следует проверять доступными статистическими методами [см. Снедекор и Кокрен (1967)].

2.7.3. Регрессионный анализ

Когда все факторы в эксперименте являются количественными, можно построить эмпирическую модель, оценивающую индекс производительности не в терминах компонентов общей дисперсии, а как функцию уровней факторов. Например, в эксперименте над системой со страничной организацией памяти, описанном в примерах 2.7 и 2.8, рассмотрим только факторы M и I, которые можно выразить количественно, и выберем фиксированные уровни для алгоритма замещения и для порядка подпрограмм. Возможно, мы захотим выразить число страничных отказов, порожденных компилятором ФОРТРАНа, как функцию от размера основной памяти и от длины компилируемой программы. Такая модель, называемая *регрессионной моделью*, может быть использована (с осторожностью) как средство прогнозирования количества страничных отказов, соответствующих уровням M и I, не рассмотренным в эксперименте.

Наше обсуждение применения регресионного анализа к оценке вычислительных систем основано на работе Вальдбаума (1973).

Пример 2.11. Нужно определить влияние определенных изменений на производительность диалоговой операционной системы APL, работающей под управлением OS/MVT в машине IBM/360 модели 91. Рассматриваются следующие изменения:

a) увеличение максимального размера *рабочей области* для процесса APL с 40 000 байт (5 дорожек на диске) до 48K байт (7 дорожек на диске);

b) уменьшение от 3 до 2 максимального количества рабочих областей, которым разрешается одновременно находиться в основной памяти; это уменьшение необходимо, чтобы избежать роста области APL в основной памяти, что оказало бы влияние на остальных пользователей системы (не принадлежащих APL).

Выбранный индекс производительности — *время реакции системы*, определенное как продолжительность интервала между моментом обнаружения системой символа, завершающего вводимую строку, и моментом начала использования центрального процессора соответствующим процессом. В изучаемой установке большая часть команд обрабатывается менее чем за один квант времени, что, вообще говоря, слишком мало по сравнению с временем реакции. Таким образом, этот индекс производительности, который можно использовать для оценки эффективности планировщика, очень мало отличается от времени ответа.

В табл. 2.20 перечислены факторы, выбранные для эксперимента. Каждый из двух первичных факторов X_1 и X_2 имеет два уровня, упомянутые выше в «a» и «b». Эксперимент должен выполняться при производственной рабочей нагрузке. Следовательно, для описания рабочей нагрузки должны быть введены некоторые вторичные факторы, не управляемые, но наблюдаемые, изменчивость которых можно уменьшить, собирая данные каждый день всегда в одно и то же время дня. Вальбаум отобрал шесть вторичных факторов, перечисленных в табл. 2.20. Был принят план «изменение факторов по одному» с несколькими наблюдениями на одну комбинацию уровней X_1 и X_2. Количество наблюдений, проведенных Вальдбаумом для каждой комбинации, приведено в табл. 2.21. В каждом сеансе мы наблюдаем распределение

Таблица 2.20

Факторы и индексы производительности в эксперименте, рассмотренном в примере 2.11

Первичные факторы

X_1 = число рабочих областей в основной памяти (два уровня: 2, 3)

X_2 = максимальное число дорожек на диске на одну рабочую область (два уровня: 5, 7)

Вторичные факторы

X_3 = число диалоговых реплик в час

X_4 = коэффициент использования ЦП для запросов с временем ЦП $\leqslant 2$ с

X_5 = коэффициент использования ЦП для запросов с временем ЦП > 2 с

X_6 = число запросов в час с временем ЦП > 2 с

X_7 = число команд в час, требующих две рабочие области в основной памяти одновременно

X_8 = число вхождений в систему в час

Индексы (точки распределения времени реакции системы t_r)

Y_1 = доля вход. сообщ. с $t_r \leqslant 0.5$ с

Y_2 = доля вход. сообщ. с $t_r \leqslant 1.0$ с

Y_3 = доля вход. сообщ. с $t_r \leqslant 1.5$ с

Y_4 = доля вход. сообщ. с $t_r \leqslant 2.0$ с

Y_5 = доля вход. сообщ. с $t_r \leqslant 3.0$ с

Y_6 = доля вход. сообщ. с $t_r \leqslant 4.0$ с

Y_7 = 50-я процентиль от t_r (медиана)

Y_8 = 90-я процентиль t_r

Y_9 = 95-я процентиль t_r

Источник: Вальдбаум (1973).

времени реакции системы Y, которое для целей данного исследования характеризуется координатами девяти точек $Y_1, \ldots, Y_9$, перечисленных в табл. 2.20, и соответствующими значениями шести факторов рабочей нагрузки.

Так как все факторы количественные, для построения эмпирической модели можно использовать регрессионные методы. Это обычно делается в три этапа, которые должны циклически повторяться до тех пор, пока модель не будет удовлетворительно аппроксимировать экспериментальные данные:

a) Выбирается математическая форма для выражения соотношения между зависимыми переменными (индексы производительности) и независимыми переменными (факторы);

b) *параметры регрессии*, имеющиеся в модели (предложенной на этапе «a»), вычисляются по экспериментальным данным;

c) оценивается качество аппроксимации экспериментальных данных моделью.

Регрессионная модель есть набор уравнений, каждое из которых связывает одну зависимую переменную с v независимыми переменными и $r+1$ параметрами регрессии (их надлежит определить на этапе «b»). Например, модель для Y_g $(g=1, \ldots, 9)$

Таблица 2.21

План эксперимента в примере 2.11

		X_2	
		5 дорожек (36 000 байт)	7 дорожек (48K байт)
X_1	2	9 дней	8 дней
	3	19 дней	—

Источник: Вальдбаум (1973).

в данном эксперименте имеет общую форму

$$Y_g = Y_g(X_1, \ldots, X_v; \beta_{g0}, \beta_{g1}, \ldots, \beta_{gr}) + E_g, \tag{2.48}$$

где $v=8$, а E_g — ошибка регрессии. Наблюдение Y_g есть набор из $(v+1)$ значений $(Y_g, X_1, \ldots, X_v)$. Пусть n — общее количество наблюдений (в нашем примере $n=$ 30). Тогда модель (2.48) должна быть применима к каждому из них:

$$Y_{gk} = Y_g(X_{1k}, \ldots, X_{vk}; \beta_{g0}, \beta_{g1}, \ldots, \beta_{gr}) + E_{gk} \quad (k=1, \ldots, n). \tag{2.49}$$

Выбор модели может диктоваться априорным знанием математической формы соотношений или интуицией исследователя. Очень часто первоначально предполагается *линейная модель*, так как линейная регрессия проще и лучше изучена, чем нелинейная. Линейность регрессии относится к соотношению между зависимой переменной и параметрами. Поэтому линейная модель есть полиномиальная форма от независимых переменных. В частности, линейная модель *первого порядка* для Y_g в нашем примере есть

$$Y_g = \beta_{g0} + \sum_{i=1}^{v} \beta_{gi} X_i + E_g. \tag{2.50}$$

Заметим, что в линейной модели первого порядка мы имеем $r=v$.

Существуют различные методы оценки параметров регрессии. Наиболее широко в линейной регрессии используется *метод наименьших квадратов*, который, по теореме Гаусса [см. Дрейпер и Смит (1966)], дает для каждого параметра несмещенную оценку, обладающую минимальной дисперсией среди всех возможных оценок, являющихся линейными функциями наблюдений $Y_{g1}, \ldots, Y_{gn}$. Оценка наименьших квадратов минимизирует сумму квадратов ошибок:

$$\sum_{k=1}^{n} E_{gk}^2 = \sum_{k=1}^{n} \left(Y_{gk} - \beta_{g0} - \sum_{k=1}^{v} \beta_{gi} X_{ik} \right)^2. \tag{2.51}$$

Оценки для параметров $\beta_{g0}, \ldots, \beta_{gv}$ получаются дифференцированием формулы (2.51) по каждому из них, приравниванием производных нулю и решением полученных $(v+1)$ алгебраических уравнений. Чтобы показать, что полученные таким способом величины являются оценками параметров регрессии, обозначим их через $b_{g0}, \ldots, b_{gv}$, где $b_{gi} = \hat{\beta}_{gi}$ $(i=1, \ldots, v)$. *Уравнение регрессии* Y_g может быть записано в виде

$$\hat{Y}_g = b_{g0} + \sum_{i=1}^{v} b_{gi} X_i, \tag{2.52}$$

откуда имеем

$$\hat{Y}_{gk} = b_{g0} + \sum_{i=1}^{v} b_{gi} X_{ik} \quad (k=1, \ldots, n). \tag{2.53}$$

Качество аппроксимации этой модели можно оценить различными способами. Например, можно использовать дисперсионный анализ для разложения общей дисперсии наблюдений Y_g на член, объясняемый уравнением регрессии, и остаточный член, обусловленный отклонениями наблюдений от модели, т. е. ошибками:

$$\sum_{k=1}^{n} (Y_{gk} - \bar{Y}_{g.})^2 = \sum_{k=1}^{n} (\hat{Y}_{gk} - \bar{Y}_{g.})^2 + \sum_{k=1}^{n} (Y_{gk} - \hat{Y}_{gk})^2, \tag{2.54}$$

где $\bar{Y}_{g.}$ — среднее всех наблюдений Y_g (общее среднее).

Таблица дисперсионного анализа, соответствующая суммам квадратов в (2.54), приведена в виде табл. 2.22. Значимость регрессии может быть проверена при помощи

F-теста, описанного в разд. 2.7.2, при условии что ошибки — взаимно независимые, нормально распределенные случайные переменные с нулевым средним и одинаковыми дисперсиями σ_E^2. Тогда гипотеза

$$H_0: \beta_{g1} = \beta_{g2} = \ldots = \beta_{gv} = 0 \tag{2.55}$$

отвергается на уровне значимости α, если

$$\frac{MSR}{MSE} > F_{\alpha;\, r,\, n-r-1}. \tag{2.56}$$

Если H_0 отвергнута, регрессия значима на уровне α. Это означает, что доля общей дисперсии, объясняемая таким уравнением регрессии, как (2.52), больше, чем ожидалось в 100 $(1-\alpha)$% подобных наблюдений с тем же самым n.

Однако то, что H_0 отвергнута, еще не означает, что все члены в модели значимы. Рассмотрим модель первого порядка типа (2.52). Для испытания значимости отдельного фактора X_i можно выполнить частичный F-тест для параметра β_{gi}. Вычисляются

Таблица 2.22

Дисперсионный анализ Y_g в примере 2.11

Источник изменчивости	Сумма квадратов	Степени свободы	Средний квадрат
Регрессия R	$SSR = \sum_{k=1}^{n} (\hat{Y}_{gk} - \overline{Y}_{g.})^2$	r	$MSR = SSR/r$
Ошибка E	$SSE = \sum_{k=1}^{n} (Y_{gk} - \hat{Y}_{gk})^2$	$n-r-1$	$MSE = SSE/(n-r-1)$
Общее T	$SST = \sum_{k=1}^{n} (Y_{gk} - \overline{Y}_{g.})^2$	$n-1$	

параметры другого уравнения регрессии, совпадающего с рассмотренным во всем, за исключением того, что член с X_i отсутствует. Сумма квадратов регрессии для этой модели SSR' имеет $r-1$ степеней свободы. Разность $SSR_i = SSR - SSR'$ имеет одну степень свободы, а ее средний квадрат равен $MSR_i = SSR_i$. Таким образом, гипотеза $H_0^i: \beta_{gi} = 0$ отвергается на уровне значимости α, если

$$\frac{MSR_i}{MSE} > F_{\alpha;\, 1,\, n-r-1}. \tag{2.57}$$

Если H_0^i нельзя отвергнуть, член X_i может быть отброшен.

Частичный F-тест проверяет значимость члена, как если бы этот член был добавлен к модели последним. Если уравнение регрессии строится последовательным добавлением членов, статистическую значимость каждого члена можно проверить *последовательным F-тестом*, который основан на процедуре, описанной выше для частичных F-тестов.

Другой мерой качества аппроксимации для уравнения регрессии является квадрат *множественного коэффициента корреляции*, $R^2 = SSR/SST$. Как мы уже видели в разд. 2.7.2, это отношение представляет часть общей дисперсии относительно среднего, объясняемую уравнением регрессии. Если $R^2 = 1$, уравнение регрессии может полностью предсказать данные ($SSE = 0$). Если $R^2 = 0$, мы имеем $SSR = 0$. Тогда в силу (2.54) $\hat{Y}_{gk} = \overline{Y}_g$. для всех k и в силу (2.52) $\hat{Y}_g = \overline{Y}_{g.} = b_{g0}$, $b_{gi} = 0$ $(i = 1, \ldots, v)$.

Таким образом, R^2 является мерой полезности членов, содержащих факторы модели: чем ближе R^2 к единице, тем лучше аппроксимация.

Значения параметров регрессии в (2.52) и соответствующие значения R^2, найденные Вальдбаумом (1973), приведены в табл. 2.23. Так как значения R^2 для девяти уравнений регрессии недостаточно близки к 1, нужно увеличить точность модели. Вальдбаум повторил анализ для *модели второго порядка*, или *квадратичной линейной модели*

$$Y_g = \beta_{g0} + \sum_{i=1}^{v} \beta_{gi} X_i + \sum_{i=1}^{v} \sum_{j=1}^{v} \beta_{gij} X_i X_j + E_g. \tag{2.58}$$

Для регрессионной модели чем больше членов в уравнении, тем лучше аппроксимация. Однако стоимость сбора и обработки данных для большого числа факторов v (если уравнения регрессии будут использованы для прогнозирования) такова, что v

Таблица 2.23

Оценка наименьших квадратов регрессионных параметров и квадрата коэффициента множественной корреляции для модели первого порядка в (2.59)

Индекс	b_{g0}	b_{g1}	b_{g2}	b_{g3}	b_{g4}	b_{g5}	b_{g6}	b_{g7}	b_{g8}	R^2
Y_1	135.7	1.076	−6.349	0.00471	8.512	−0.5827	−0.1668	−0.1469	−0.8859	0.70
Y_2	115.5	0.309	−2.602	0.00382	3.349	−0.2329	−0.0604	−0.1036	−0.1683	0.67
Y_3	106.4	0.036	−1.292	0.00288	1.227	−0.0830	−0.0358	−0.0579	−0.0703	0.63
Y_4	102.6	0.018	−0.718	0.00218	0.477	−0.0330	−0.0287	−0.0328	−0.0674	0.60
Y_5	100.1	−0.016	−0.337	0.00161	0.121	−0.0111	−0.0156	−0.0183	−0.0464	0.52
Y_6	99.1	0.017	−0.202	0.00139	0.042	−0.0044	−0.0119	−0.0135	−0.0345	0.47
Y_7	−0.0770	−0.0183	0.0435	−0.00002	−0.051	0.0033	0.0008	0.0009	0.0056	0.74
Y_8	−0.9581	−0.0136	0.2029	−0.00026	−0.264	0.0178	0.0039	0.0072	0.0146	0.68
Y_9	−1.660	−0.0346	0.3242	−0.00066	−0.292	0.0242	0.0082	0.0123	0.0387	0.61

Источник: Вальдбаум (1973)

следует минимизировать. Следовательно, должен быть достигнут компромисс, состоящий во включении в модель только тех членов, которые оказывают существенное влияние на индекс производительности. Существуют различные интерационные процедуры выделения этих членов помимо исчерпывающего подхода, рассматривающего все возможные уравнения регрессии с последующим выбором уравнения, обеспечивающего наилучший компромисс между противоположными требованиями экономии и качества аппроксимации [см., например, Дрейпер и Смит (1966)].

Процедура *вычеркивания отстающих* проверяет с помощью частичного F-теста значимость каждого члена в полном уравнении регрессии, содержащем все члены, например в (2.58). Наименее значащий член среди не удовлетворяющих тесту вычеркивается. Каждый раз, когда вычеркивается некоторый член, вычисляется новая аппроксимирующая модель, и процедура завершается, когда все члены удовлетворяют частичному F-тесту.

Процедура *отбора ведущих* начинается с простейшего уравнения $Y_g = Y_g(X_i)$, где X_i — переменная, наиболее коррелированная с Y_g. Затем добавляются члены в порядке убывания частного коэффициента корреляции каждого члена с Y_g до достижения удовлетворительного уравнения регрессии. Метод вычисления частных коэффициентов корреляции описан Дрейпером и Смитом (1966).

Процедура *пошаговой регрессии* является модификацией процедуры отбора ведущих. Каждый раз, когда добавляется новый член, проверяется значимость ранее введенных членов, и те члены, вклад которых незначим, вычеркиваются. Процедура прекращается, когда нельзя ни добавить, ни вычеркнуть ни одного члена.

Можно использовать также другие процедуры, некоторые из которых являются смесями описанных выше. Например, Вальдбаум использовал сочетание процедуры

вычеркивания отстающих с пошаговой регрессией. Оказались вычеркнутыми все члены квадратичной модели (2.58), кроме шести:

$$\hat{Y}_g = b_{g0} + b_{g33}X_3^2 + b_{g77}X_7^2 + b_{g24}X_2X_4 + b_{g27}X_2X_7 + b_{g47}X_4X_7 + b_{g58}X_5X_8 \quad (g=1, \ldots, 9). \quad (2.59)$$

Значения параметров и R^2 для модели (2.59) приведены в табл. 2.24. Значение R^2 оказывается существенно выше, чем в модели первого порядка, которая включала восемь членов вместо шести. Было обнаружено, что уравнения для $\bar{Y}_5$ и $\bar{Y}_6$ являются значимыми на уровне 5%, а остальные уравнения регрессии — на уровне 1%.

Таблица 2.24

Оценки наименьших квадратов параметров регрессии и квадратичного коэффициента множественной регрессии для модели второго порядка в (2.59)

Индекс	b_{g0}	$10^7 \times b_{g33}$	$10^4 \times b_{g77}$	b_{g24}	b_{g27}	b_{g47}	b_{g58}	R^2
Y_1	107.2	13.2	39.9	7.155	−0.1441	−0.2568	−0.0341	0.82
Y_2	105.8	12.8	18.6	3.622	−0.0686	−0.1348	−0.0135	0.83
Y_3	101.6	9.5	9.34	1.851	−0.0346	−0.0722	−0.00531	0.81
Y_4	99.7	6.88	5.28	1.030	−0.0193	−0.0418	−0.00255	0.74
Y_5	98.6	4.91	2.54	0.528	−0.00959	−0.0222	−0.00115	0.60
Y_6	98.4	4.28	1.69	0.3667	−0.00653	−0.0156	−0.00068	0.52
Y_7	0.089	−0.046	−0.23	−0.0387	0.000884	0.00134	0.000179	0.82
Y_8	−0.172	−0.873	−1.40	−0.2686	0.00513	0.00984	0.00103	0.86
Y_9	−0.257	−2.06	−2.53	−0.4876	0.00891	0.01942	0.00151	0.81

Источник: Вальдбаум (1973)

Знание параметров, приведенных в табл. 2.23 и 2.24, позволяет предсказать распределение времени реакции системы для данной рабочей нагрузки и данных уровней первичных факторов. Например, рассмотрим рабочую нагрузку, характеризующуюся средними значениями 6 вторичных факторов по 36 проведенным сеансам: $X_3=1505$ сообщений в диалоге за один час; $X_4=1.48\%$; $X_5=12.87\%$; $X_6=18.9$ больших запросов на центральный процессор в час; $X_7=120$ команд, требующих две рабочие области в памяти в течение часа; $X_8=19.7$ вхождений в режим диалога в течение часа. Используя модель первого порядка (2.52) и параметры табл. 2.23, получим оценки медианы распределения времени реакции Y_7:

a) для $X_1=2$ рабочие области, $X_2=5$ дорожек; $\hat{Y}_7=0.2742$ с;
b) для $X_1=2$ рабочие области, $X_2=7$ дорожек; $\hat{Y}_7=0.3612$ с;
c) для $X_1=3$ рабочие области, $X_2=5$ дорожек; $\hat{Y}_7=0.2559$ с;
d) для $X_1=3$ рабочие области, $X_2=7$ дорожек; $\hat{Y}_7=0.3429$ с.

Таким образом, очевидно, что значение X_1 влияет слабо, тогда как значение X_2 оказывает довольно большое влияние на Y_7. Увеличение размера рабочей области вызывает существенное увеличение времени реакции, соответствующего медиане распределения. Для выбранной рабочей нагрузки вклады членов X_7 и X_8 являются наибольшими после члена X_2. Физическая интерпретация этих результатов (и других), как это часто оказывается в регрессионных моделях, довольно затруднительна. Эти модели действительно полезны для анализа данных, но не имеют отношения к внутренней структуре и действию изучаемой системы. Следовательно, объяснение результатов регрессионного анализа в терминах системы часто оказывается нелегкой задачей.

Если мы воспользуемся квадратичной моделью (2.59) и параметрами из табл. 2.24, то получим

a) для $X_2=5$ дорожек: $\hat{Y}_7=0.2747$ с;
b) для $X_2=7$ дорожек: $\hat{Y}_7=0.3723$ с.

Членами с наибольшим влиянием в порядке уменьшения влияния на $\hat{Y}_7$ являются X_2X_5, X_2X_4, X_7^2, X_4X_7. Из них X_2X_4 и X_7^2 отрицательны. Чувствительность $\hat{Y}_7$ к различным факторам можно оценить, вычислив производную от (2.59) по тому или иному фактору. Это вычисление просто, если предполагается, что факторы взаимно независимы.

Двумя наиболее примечательными аспектами этой квадратичной модели являются отсутствие членов первого порядка и то, что X_1, один из двух первичных факторов, не появляется в уравнении регрессии. Мы уже видели выше, что даже в модели первого порядка изменения X_1 не оказывали достаточно большого влияния на $\hat{Y}_7$.

Вальдбаум сравнил некоторые предсказания, полученные с помощью квадратичной модели, с результатами последующих измерений; согласие оказалось вполне удовлетворительным. ■

Задачи

2.1. Сконструируйте два фиксированных аппаратных измерителя, дающих на своих выходах электрический сигнал, пропорциональный измеряемой величине. Оба измерителя должны использоваться в диалоговой системе: один — для измерения коэффициента использования центрального процессора, а другой — средней пропускной способности. Измерения производятся во взаимодействиях в секунду на коротком интервале, предшествующем данному моменту времени. Допустим, что при передаче нового выходного сообщения на терминал на доступной линии возникает импульс. Сигнал, создаваемый этими измерителями, может быть (но не обязательно) уровнем напряжения постоянного тока. Это должно использоваться для визуальных (механических или оптических) индикаторов этих двух величин, показания которых считываются операторами машины.

2.2. Модифицируйте схему эксперимента на рис. 2.7 так, чтобы измерять в ПРОП-системе, рассмотренной в примере 2.2, следующие величины:

(a) система простаивает,
(b) перекрытие «центральный процессор занят/канал занят»,
(c) центральный процессор простаивает,
(d) канал занят.

Достаточно ли значений этих четырех величин для определения значений остальных величин в табл. 2.3?

2.3. В вычислительной системе, используемой для работы в реальном времени, важно определить задержку, с которой центральный процессор обслуживает определенные прерывания. Для этих прерываний мы хотим измерить интервал времени, между их поступлением и началом выполнения их обслуживающих программ. Предположим, что центральный процессор имеет микропрограммный блок управления. Какие типы измерителей можно использовать: аппаратный, микропрограммный, программный? Каким типом измерителя вы бы воспользовались? Почему? Как? Можно ли использовать выборочный измеритель? Как?

2.4. Максимальная полоса пропускания памяти произвольного доступа есть максимальное число слов, которые можно прочесть из памяти или записать в нее за единицу времени. Для простоты предположим, что полоса пропускания чтения равна полосе пропускания записи. Реальная полоса пропускания этой же памяти равна числу слов, которые действительно передаются в память или из памяти в единицу времени. Максимальная полоса пропускания входит в цели конструирования, а для данной памяти представляет собой известную постоянную верхнюю границу для реальной полосы. Наоборот, реальная полоса пропускания вообще неизвестна и меняется во времени, так как она зависит, среди прочего, от интенсивности запросов, поступающих от различных блоков системы. Отношение «реальная полоса/максимальная по-

лоса» является индексом использования полосы пропускания памяти. Как бы вы измерили действующую полосу пропускания памяти в работающей системе? Сделав соответствующие предположения о структуре памяти и доступных сигналах, постройте схему эксперимента типа показанной на рис. 2.7 для иллюстрации, как можно было бы использовать аппаратный измеритель с фиксированной программой. Кроме того, сконструируйте аппаратный измеритель с фиксированной программой для той же цели, определив заранее, как будут предъявлены экспериментатору результаты измерений. Можно ли использовать программный измеритель? Если да, то какого типа и как?

2.5. Необходимо измерить коэффициент использования основной памяти в вычислительной системе. Для этой цели память размером $128K$ делится на 128 областей по $1K$ слов каждая. Время эксперимента делится на интервалы длительности τ. Типичные значения τ лежат в интервале между 1 мс и 1 с. Булева переменная $a_j(t)$ используется для описания поведения во времени области j ($j=0, 1, \ldots, 127$). В момент t_i, являющийся концом интервала (t_{i-1}, t_i), где $t_{i-1}=t_i-\tau$, мы полагаем $a_j(t_i)=1$, если в течение (t_{i-1}, t_i) было хотя бы одно обращение к области j. В противоположном случае мы считаем $a_j(t_i)=0$. Определите, как мог бы быть использован измеритель с запоминаемой программой для сбора и накопления последовательностей значений a_j. Укажите, как можно обнаруживать ссылки на каждую область и какие связи нужно установить между измеряемым компьютером и измерителем (см., например, рис. 2.8). Можно ли использовать измеритель с постоянной программой? Если да, то какие средства потребуются?

2.6. В вычислительной системе имеется диск с подвижными головками емкостью 400 цилиндров с характеристикой поиска, аппроксимируемой уравнением $t_{\text{поиск}}=25+0.3(n_{\text{ц}}-1)$, где $t_{\text{поиск}}$ измеряется в миллисекундах, а $n_{\text{ц}}$ — число пересекаемых цилиндров. Нужны измеритель и схема эксперимента, позволяющие собирать и запоминать гистограмму $n_{\text{ц}}$ за сравнительно длинные периоды времени. Операционная система содержит справочник, из которого можно узнать адреса (цилиндр, дорожка, сектор) всех файлов, содержащихся на диске, и всех пустых областей на нем. Когда предстоит получить доступ к диску, справочник дает номер цилиндра, к которому должны быть подведены головки. Затем центральный процессор передает номер цилиндра контроллеру диска, который управляет поиском. По аналогии с табл. 2.5 задайте внутренне управляемый программный измеритель, выполняющий описанную выше задачу. Можно ли использовать для этой цели аппаратный измеритель с запоминаемой программой? Если да, то каковы его преимущества и недостатки по сравнению с программным подходом?

2.7. Можно ли использовать для измерения гистограммы $n_{\text{ц}}$ в задаче 2.6 выборочный измеритель? Почему?

2.8. Продолжите сеанс работы измерителя SMT (в табл. 2.9), переведя во включенное состояние контрольные точки, размещенные в части (b) этой таблицы и добавив дополнительные контрольные точки для измерения количества вызовов генератора псевдослучайных чисел. Контрольная точка, вызывающая измерительную программу TCNT, размещенная в части (a) таблицы, должна быть выключена. Проанализируйте искажения во времени и пространстве, создаваемые измерительным оснащением этой программы с учетом описанных выше модификаций. Укажите, как можно уменьшить эти искажения, если это необходимо, или по крайней мере как их учесть.

2.9. Являются ли, по вашему мнению, коды операций центрального процессора независимыми или нет? Сконструируйте измеритель для измерения автокорреляции, которая может существовать в последовательности кодов операций, предполагая, что биты кодов операций на регистре команд доступны, как на рис. 2.6. Определите быстродействие необходимых для этого электрических цепей, считая длительность цикла самой короткой команды равной 800 нс.

2.10. Является ли правильным решением для измерения коэффициентов использования команд центрального процессора периодическая выборка битов кодов операций из регистра команд? Почему? Если вы ответите на первый вопрос «нет», укажите, в чем именно будут неточны результаты выборочного эксперимента и какие исправления следует внести для улучшения их точности.

2.11. Необходимо построить кривую, связывающую среднее время ответа $\bar{t}$ с запросом времени центрального процессора на одно взаимодействие, для диалоговой системы. Взаимодействия подразделяются по величине запроса времени центрального процессора на десять категорий. Искомая кривая получится интерполяцией 10 экспериментальных точек. Интервалы величин запроса времени центрального процессора, соответствующие этим категориям, выбраны так, чтобы относительные частоты взаимодействий (диалогов) в каждой категории были приблизительно одинаковы. Оценка средней интенсивности взаимодействий на протяжении интересующего нас периода составляет 20 взаимодействий в минуту. Предполагая, что выборочные средние распределены вблизи истинного среднего каждой категории нормально и что стандартное отклонение σ_t времени ответа в каждой категории известно, определите минимальную длительность сеанса измерения в следующих случаях:

$\sigma_t/\bar{t}$	50%	100%	50%	200%
$\bar{e}/\bar{t}$	1%	1%	3%	5%
K	99%	95%	90%	90%

Правдоподобно ли предположение о нормальном распределении?

2.12. Опишите, как бы вы оценили дисперсию выборочных средних относительно истинных средних в экспериментах, описанных в задаче 2.11. Примените метод независимых копий.

2.13. В таблице З.2.1 приведены некоторые результаты, полученные Шветманом и Брауном (1972) в их эксперименте типа «изменение факторов по одному» на вычислительной системе CDC 6600. Спланируйте эксперимент для определения результата взаимодействия между максимальной степенью мультипрограммирования и размером основной памяти. Ожидали бы вы большего или меньшего эффекта взаимодействия, если бы уровни были выбраны в соответствии с табл. З.2.1? Какие уровни выбрали бы вы для вашего эксперимента? Зная, что средний размер задания во время эксперимента был равен $21K$ слов, можете ли вы объяснить, почему эффект уменьшения размера основной памяти был намного меньше, чем следовало бы ожидать?

Таблица З.2.1

Результаты двух однофакторных экспериментов

	Использование ЦП (%)	Средняя пропускная способность (заданий/час)
Фактор: максимальная степень мультипрограммирования		
5 (нормальная)	96	429
4	92	392
3	91	393
2	76	328
1	60	262
Фактор: размер основной памяти (K слов)		
128	97	426
120	96	417
112	93	406
104	93	406
96	93	403
88	91	400

Источник: Шветман и Браун (1972).

2.14. В диалоговой мультипрограммной системе заданиям отводятся множества смежных адресов основной памяти, которые они занимают полностью. Когда задание вынуждено ждать завершения операции ввода-вывода, оно разгружается из памяти. Когда оно снова готово к выполнению и необходимое пространство в памяти свободно, задание загружается. Если задание перемещаемо, оно может быть загружено в любое множество свободных смежных адресов основной памяти. В противном случае задание может быть загружено только тогда, когда освободится область, для которой оно было запрограммировано. Вообще говоря, пространство памяти, представленное заданию, используется только частично. Поэтому, прежде чем задание будет разгружено, операционная система может попытаться исключить неиспользуемые части пространства памяти, уменьшая тем самым количество информации, перемещаемой в память и из памяти и увеличивая вероятность отыскания пространства для загружаемого задания. Эта операция, требующая некоторых дополнительных вычислительных ресурсов и, следовательно, увеличивающая накладные расходы, была названа Нильсеном (1971) *уплотнением*. Он описал эксперимент с моделью, подобной обсуждаемому здесь. Спланируйте эксперимент по определению эффектов перемещения, уплотнения и их воздействия на производительность системы.

(a) Выберите индекс (индексы) производительности.

(b) Определите факторы (первичные и вторичные) и выберите их уровни.

(c) Подготовьте план эксперимента.

(d) Опишите методы, которыми вы воспользуетесь для интерпретации результатов. Запомните, что необходимо ответить на такие вопросы: Насколько полезно перемещение? Полезно ли уплотнение? Как они взаимодействуют?

2.15. Постройте круговую диаграмму для Y_1 (доля взаимодействий с временем реакции системы не более 0.5 с) по данным эксперимента Вальдбаума, обсужденного в примере 2.11, с учетом модели (2.59) и табл. 2.24. Круговая диаграмма должна показать долю общей дисперсии, объясняемую уравнением регрессии, и долю, отнесенную за счет ошибок. Можете ли вы на основании имеющихся данных разделить дисперсию, объясняемую регрессией, на компоненты, соответствующие первичным (X_1, X_2) и вторичным (X_3, . . . , X_8) факторам? Если ответите «да», усовершенствуйте соответственно вашу круговую диаграмму. Если «нет», укажите, каких данных вам не хватает. Независимо от ответа на предыдущий вопрос, можете ли вы вычислить вес каждого фактора в индексе производительности Y_1 (не в его дисперсии)? Если да, то определите эти веса для данной нагрузки и для всех четырех комбинаций двух уровней для X_1 и X_2, рассмотренных в примере 2.11, и постройте четыре круговые диаграммы, соответствующие этим комбинациям. Можете ли вы сделать на основании этих результатов какое-нибудь заключение относительно проблемы оценки, ради которой был запланирован этот эксперимент?

2.16. Каждый сеанс измерения состоит в прогоне фиксированного набора заданий через предназначенную для этого систему пакетной обработки и измерении общего времени обработки заданий в этом наборе. Конфигурация системы включает один центральный процессор и два канала ввода-вывода, один из которых управляет дисковым устройством с подвижными головками, а другой — рядом низкоскоростных периферийных устройств (устройство ввода перфокарт, устройство печати, лентопротяжный механизм и др.). Предположим, что два сеанса с одинаковой рабочей нагрузкой и с одной и той же системой (как аппаратной, так и программной) имеют длительности, различающиеся на 15%.

(a) Какие гипотезы могли бы вы предложить относительно возможных источников этого различия?

(b) Как бы вы проверили справедливость этих гипотез?

(c) В предположении, что одна из ваших гипотез оказалась верной, что бы вы сделали для исключения воздействия этого источника изменения на результаты сеансов вашего эксперимента?

3
Методы моделирования

3.1. РАБОТЫ ПО МОДЕЛИРОВАНИЮ

Моделирование было определено в разд. 1.6 как метод оценки, представляющий с помощью модели поведение системы на отрезке времени. Наблюдение поведения модели системы во времени под влиянием событий, генерируемых моделью входных воздействий системы, дает оцененные результаты, которые можно использовать в работах по оценке. Модель, пригодная для этой цели, называется *имитационной моделью*, или просто *моделью*.

Существенным условием применимости любой модели является ее правдоподобие. Результатам, полученным в экспериментах с моделью, нельзя доверять, если модель недостаточно точна. В контексте работы по оценке *точность* модели должна быть определена по отношению к индексу (или индексам) производительности, выбранному для этой

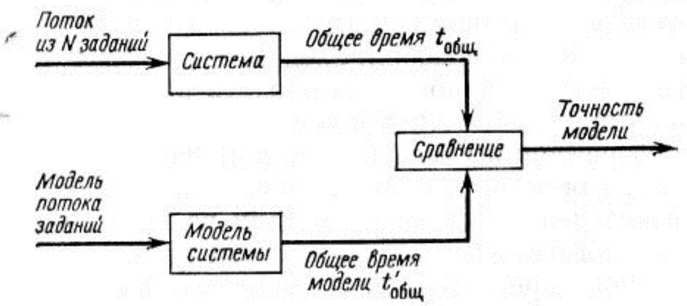


Рис. 3.1. Иллюстрация понятия точности модели.

цели. Значения этих индексов, полученные в эксперименте на модели, должны быть достаточно близки к значениям, которые дала бы моделируемая система при тех же входных воздействиях.

Рис. 3.1 иллюстрирует это определение для простого случая. Рассматриваемая система — система пакетной обработки; ее рабочая нагрузка — поток из N заданий. Если индекс производительности, который нас интересует,— это общее время $t_{общ}$ обработки N заданий, то модель считается точной, когда $|t_{общ}-t'_{общ}|<e$, где e — заданная максимальная ошибка, а $t'_{общ}$ — результат моделирования. Разумеется, это определение предполагает, что $t_{общ}$ можно измерить с достаточной точностью. Заметим, что имитационная модель иногда дает не непосредственное значение $t'_{общ}$, а величину, которую можно преобразовать в $t'_{общ}$ подходящими действиями, например умножением на константу.

Имитационную модель можно рассматривать как некую систему, особенно если она воспроизводит не только поведение моделируемой системы, но также ее структуру и организацию. Это воспроизведение следует некоторым *правилам подобия*, очень сильно зависящим от природы модели. При применении этих правил большая часть математических соотношений между переменными, которые могут использоваться для описания системы, может остаться скрытой в неявной форме в модели, как они скрыты в системе.

В некоторых других случаях модель состоит из математических уравнений, описывающих динамическое поведение системы. В этом случае метод можно назвать имитационным моделированием, только если уравнения решаются численным методом, который воспроизводит эволюцию их решений на интервале времени. Таким образом, модель включает как уравнения, так и алгоритм для их решения. В терминах, введенных в разд. 1.6, математическая модель является аналитической, если она решается любым методом, кроме имитационного моделирования. Если в процессе ее решения используется имитационное моделирование, то она будет имитационной моделью.

Упомянутому выше различию между двумя типами моделей не следует придавать слишком большого значения. Действительно, математическая структура, которая явно сформулирована в модели из уравнений, просто скрыта, но все же присутствует в модели, воспроизводящей поведение компонентов системы и их взаимодействие в соответствии с правилами подобия. Следует также заметить, что в одну и ту же модель могут одновременно входить подмодели обоих типов. Например, некоторые части или уровни системы могут моделироваться точными математическими уравнениями, а другие — с использованием правил подобия. Кроме того, во всех случаях *время модели*, с которым соотносятся наблюдения поведения модели во время моделирования, не совпадает с *реальным временем*, в котором развивается процесс моделирования. Даже время в модели, моделирующее время реальной системы, подчиняется одному из нескольких правил подобия.

В этой главе мы считаем, что, если явно не оговорено обратное, модель по структуре похожа на оцениваемую систему. В работах по оценке производительности вычислительных систем такие модели (в том числе имитационные модели, воспроизводящие математические модели, см. гл. 4) используются гораздо чаще, чем модели, состоящие из уравнений. В них можно идентифицировать *внутренние параметры*, соответствующие параметрам системы, *входные параметры*, соответствующие параметрам рабочей нагрузки, и *выходные параметры*, соответствующие первичным и вторичным индексам производительности, представляющим интерес для оценщиков. Так как модели вычислительных систем обычно воплощаются в виде программ, эти параметры в действительности будут представлены переменными (в более общем случае — структурами данных) или, если они не числовые, подпрограммами. Благодаря гибкости программ, мы можем конструировать параметрические модели, входные переменные которых включают

представление не только параметров рабочей нагрузки, но и параметров системы, так что значения этих параметров легко изменять, имитируя изменения как системы, так и рабочей нагрузки.

Вычислительные системы обычно моделируются как дискретные системы, характеризуемые дискретными состояниями и переходами

Рис. 3.2. Концептуальные модели (a) измерительного подхода и (b) имитационного подхода к задаче, рассматриваемой в примере 3.1.

между состояниями *(событиями)*, происходящими в дискретные моменты времени. Часто можно показать, что скрытая математическая структура этих моделей состоит из алгебраических и логических соотношений. Для иллюстрации этого утверждения и понятий, введенных в данном разделе, рассмотрим простой пример (см. также пример 1.3).

Пример 3.1. Требуется определить время выполнения данной программы, выполняющейся в однопрограммном режиме в ПРОП-системе (см. рис. 1.2). Предположим, что во время выполнения программа пользуется только барабаном и диском

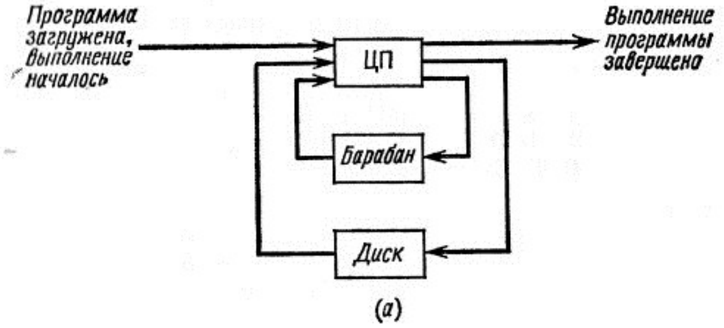


Рис. 3.3. (a) Простая блок-схема ПРОП-системы для примера 3.1; (b) — соответствующая диаграмма состояний.

и никакими другими периферийными устройствами. Концептуальная модель, лежащая в основе измерительного подхода к этой проблеме, изображена на рис. 3.2(a). Концептуальная модель, описывающая имитационный подход, представлена на рис. 3.2(b). Блок-схема на рис. 3.3(a) может быть использована в качестве основы для конструирования подходящей имитационной модели для данной задачи. Если программа полностью загружена в память, центральный процессор сразу же приступает к ее обработке. Выполнение прерывается, как только программа обращается к

барабану или диску, и возобновляется, когда запрос удовлетворен. Мы определяем время выполнения программы как интервал между началом и окончанием выполнения этой программы. Очень простая дискретная модель этой системы основывается на рассмотрении трех объектов, представленных на рис. 3.3(a), как находящихся в каждый момент времени в одном из двух состояний: *свободен* или *занят*. Если состояние центрального процессора представлено булевой переменной a_1, состояние барабана — переменной a_2, а диска — a_3, то состояние системы будет определено тройкой (a_1, a_2, a_3). Нас будут интересовать только четыре из восьми возможных состояний, как показано в табл. 3.1 и на диаграмме рис. 3.3(b). Модель этой программы, которая будет подробно рассмотрена в примере 3.3, должна содержать информацию, позволяющую имитационной модели определять моменты времени, в которые программа выдает запросы на барабан, или диск, и момент окончания программы. Разумеется, эти времена должны отсчитываться относительно начала выполнения программы. Начало выполнения, запросы на диск и барабан, завершение обслуживания диска и барабана

Таблица 3.1

Символы и обозначения, использованные в примере 3.1 и на рис. 3.3 (b)

Символ				Описание
(a) Переменные состояний				
a_1				Состояние ЦП (1: занят; 0: свободен)
a_2				Состояние барабана (1: занят; 0: свободен)
a_3				Состояние диска (1: занят; 0: свободен)
(b) Состояния системы	a_1	a_2	a_3	
S_0	0	0	0	Система свободна
S_1	1	0	0	ЦП занят
S_2	0	1	0	Барабан занят
S_3	0	0	1	Диск занят
(c) Входные события				
нв				Начало выполнения
бр				Запрос на барабан
дс				Запрос на диск
зп				Запрос удовлетворен
оп				Окончание программы
(d) Условное выражение $(X, ev) \longrightarrow Y$				Если система находится в состоянии X и произойдет входное событие ev, то состояние системы станет Y

и окончание программы — все это входные события, обусловливающие переход системы от одного состояния к другому. Следующие логические условия можно получить из диаграммы состояний рис. 3.3(b) с использованием символов и обозначений, объясненных в табл. 3.1:

$$\begin{aligned} &(S_0, \text{нв}) \longrightarrow S_1 \quad (S_2, \text{зп}) \longrightarrow S_1 \\ &(S_1, \text{бр}) \longrightarrow S_2 \quad (S_2, \text{зп}) \longrightarrow S_1 \\ &(S_1, \text{дс}) \longrightarrow S_3 \quad (S_1, \text{оп}) \longrightarrow S_0. \end{aligned} \tag{3.1}$$

Время выполнения программы совпадает с общим временем, проведенным системой в состояниях S_1, S_2 и S_3. Это время можно вычислить, построив подходящее имитационное представление системы и программы, а также алгоритма, управляющего этими двумя моделями, для воспроизведения пути программы через систему. Логиче-

ские условия (3.1) используются для определения следующего шага по этому пути. Время исполнения программы вычисляется приращениями, путем прибавления к текущему значению времени модели длительности пребывания системы в очередном состоянии. Это эквивалентно переводу часов модели на момент следующего события. Определение времени выполнения является примером роли алгебраических соотношений в модели. В разд. 3.2 мы опишем, как можно построить имитационную модель. ■

В конструировании компьютеров методы имитационного моделирования используются не только для оценки производительности различных систем, но также для проверки их правильности. Например, логические модели позволяют разработчикам аппаратуры проверить

Таблица 3.2

Характеристики двух коммерческих программ для моделирования вычислительных систем

	Модель	
	SCERT	CASE
Поставщик	COMPRESS, филиал Comten	Tesdata Systems Corp.
Выпуск первой версии	1962	1969
Минимальная память	110 K байт (основная)	256K байт (основная) 1M байт (внешняя)
Содержимое библиотеки	Стоимостные спецификации производительности всех изделий (аппаратура и программы) фирм ASI, Burroughs, CDC, Datasaab, DEC, Honeywell, IBM, ICL, NCR, Philips, Siemens, Univac, Xerox, а также некоторых изделий других фирм; библиотека обновляется ежемесячно	Параметры аппаратуры и программного обеспечения (языковые процессоры, программы сортировки, операционные системы) почти всех основных систем, а также параметры отдельных терминалов, дисплеев, модемов; библиотека обновляется ежемесячно или ежеквартально.
Входные данные	Определение данных (описание каждого файла), процедуры (описание каждой программы), конфигурация (описания оборудования и системных программ), среда (описание административной структуры)	Спецификации файлов, спецификации прогонов (файлы и операции, участвующие в каждом прогоне), спецификации конфигурации (описания оборудования и программного обеспечения)
Выходные отчеты	25 форм в 4 категориях: общие (13), мультипрограммные (3), реального времени (7), разделения времени (2); области применения отчетов: оценка управления (6), разработка и изготовление систем (17), анализ производительности (19)	Около 45 в 3 категориях: последовательная обработка, мультипрограммирование, реальное время; обеспечиваются сводные и детальные отчеты (по различным аспектам производительности систем)

правильность решений цифровых узлов или изучить влияние различных отказов на функции, выполняемые логической сетью, и на эффективность механизмов обнаружения и исправления ошибок. Хотя многие из наших соображений применимы и к моделям, созданным для

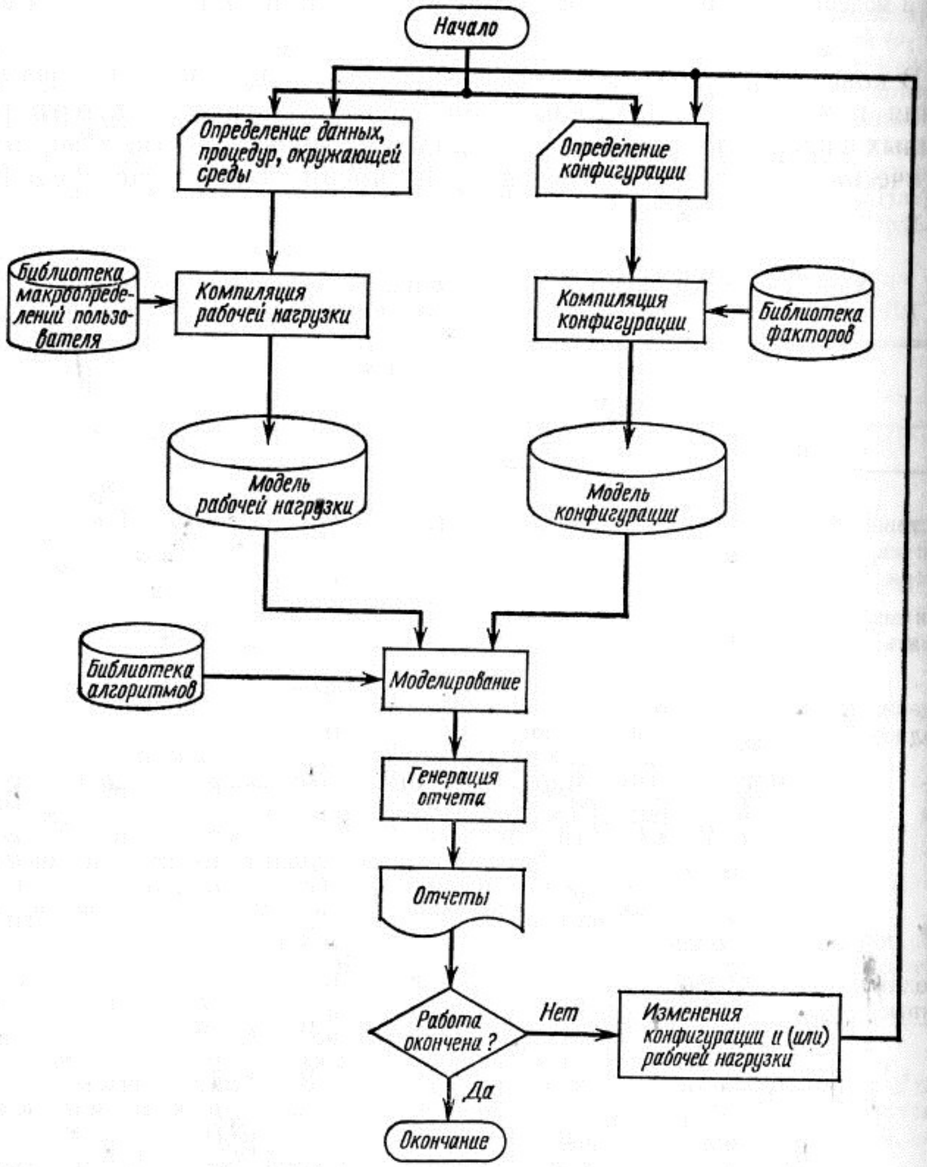


Рис. 3.4. Блок-схема итеративного исследования по оценке производительности, основанного на системе SCERT.

этих целей, мы сосредоточимся на модельном подходе к оценке производительности вычислительных систем.

Если для работы по оценке выбран метод имитационного моделирования, этап III (подготовка плана) включает *формулировку* и *построе-*

ние модели (которые будут обсуждаться в разд. 3.2), ее *калибровку и проверку пригодности (контроль)* (разд. 3.3), а также *планирование имитационных экспериментов* (которое обсуждается в разд. 3.4). *Интерпретация результатов моделирования*, кратко рассмотренная в разд. 3.5, будет выполняться на этапе V работы. Тщательная оценка всех входящих затрат (время программирования, время отладки, стоимость машинного времени) особенно необходима в работах с использованием имитационного моделирования, которое часто может оказаться недопустимо дорогим.

Следует отметить, что формулировки и конструирования модели можно избежать, используя коммерческие пакеты имитационных программ. Примерами таких моделей вычислительных систем являются

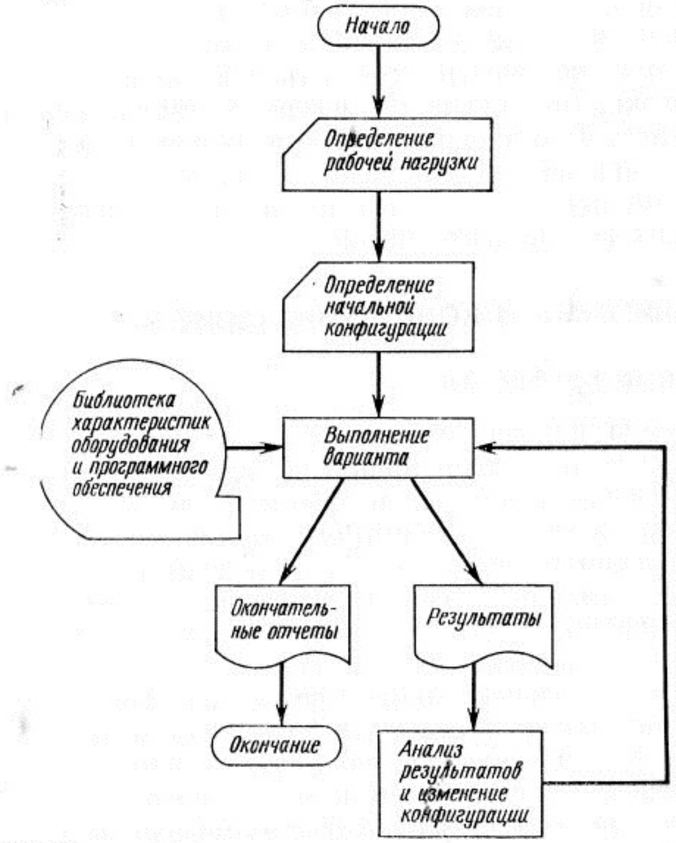


Рис. 3.5. Блок-схема исследования, выполняемого с помощью CASE.

SCERT (оценки систем и компьютеров и метод обзора) и CASE (оценка систем с помощью компьютеров). Некоторые важные особенности этих двух изделий отражены в табл. 3.2. Их блок-схемы представлены на рис. 3.4 и 3.5. Использование коммерческих моделей отнюдь не просто. Подготовка входных данных для них и интерпретация их результатов требуют работы квалифицированного аналитика в течение продолжительного времени, часто порядка нескольких дней. Эти модели

способны воспринимать описания рабочей нагрузки с различной степенью детальности. Это позволяет использовать модель для широкого спектра исследований с различной степенью знаний о прикладных программах. Этот спектр обычно включает сравнение с целью выбора, анализ с целью усовершенствования, работы по проверке возможности реализации, по конструированию и изготовлению, прогнозирование производительности. Точность результатов, вообще говоря, не известна, если не проведено сравнение с результатами, полученными другими методами с известной точностью. Например, изготовители системы SCERT утверждают, что типичная ошибка составляет 5%, если рабочая нагрузка специфицирована достаточно детально, и менее 10% для отдельного прогона. Результаты системы CASE оцениваются ее изготовителями так: типичная ошибка менее 10% с наибольшим случайным отклонением отдельного прогона от истинного значения до 50%. Разумеется, чем меньше точность и детальность спецификаций рабочей нагрузки, тем меньше мы доверяем результатам, полученным на этих моделях. Таким образом, коммерческие моделирующие программы лучше приспособлены для исследований, имеющих целью получение относительных (т. е. сравнительных) результатов, чем для работ, требующих абсолютных оценок индексов производительности.

3.2. ФОРМУЛИРОВКА И КОНСТРУИРОВАНИЕ МОДЕЛИ

3.2.1. Основные проблемы

Формулировка имитационной модели для работы по оценке вычислительной системы требует принятия ряда решений. От этих решений во многом зависит успех всей работы. Большая часть их должна быть продиктована целями работы (или оказывается под их влиянием). Поэтому важно в первую очередь определить эти цели. Чем детальнее и точнее они определены, тем выше вероятность успеха. Обычно очень рискованно и дорого создавать модель, способную ответить на все возможные вопросы о поведении вычислительной системы. Уже сообщалось о большом количестве неудач проектов моделирования, по-видимому, обусловленных этой слишком честолюбивой целью (которая практически эквивалентна отсутствию цели вообще).

Вычислительные системы, для которых имеет смысл проводить оценку, обычно слишком сложны, чтобы их моделировать с такой степенью подробности, какая нужна для получения от модели ответов на все возможные вопросы или на очень широкий спектр вопросов. При конструировании модели можно было бы идеально определить ее область применимости, приняв во внимание первоначальные цели исследования и дополнительные цели или подцели, которые можно предусмотреть в результате предварительного анализа задачи. К сожалению, часто оказывается очень трудно или невозможно надежно предсказать дальнейший ход работы. Таким образом, сферу применимости модели часто делают шире, чем это следовало бы из ее целей,

так что она может охватить некоторые неожиданные области исследования.

Обсудим кратко некоторые специфические вопросы, которые следует принимать во внимание при имитационном моделировании вычислительных систем.

а. *Детальность.* Выбор степени детальности модели (как предполагается в приведенных выше замечаниях, зависящий от целей) — это, вероятно, самая важная проблема, которую предстоит решить. Более детальная модель, по-видимому, будет более точной и будет иметь более широкую область применения, чем менее детальная. Однако, несомненно, ее проектирование, изготовление, проверка, документирование и использование обходятся намного дороже. Использование этих моделей в итерационных процедурах часто оказывается тягостным и неэффективным, так как их времена выполнения делают работу с ними в режиме диалога невозможной. Таким образом, при выборе степени детальности имитационной модели, разработчики должны найти точное и трудно определимое равновесие между требованиями исследования и ценой модели.

Как охарактеризовать степень детальности? Широко распространенным индексом степени детальности является *отношение реального времени к времени модели*. Это — соотношение между временем выполнения модели с данным набором входных данных и временем, которое потребуется моделируемой системе для обработки рабочей нагрузки, моделируемой этими входными данными. Например, если требуется 1 минута реального машинного времени для моделирования 3 минут времени моделируемой системы, отношение, характеризующее эту модель, равно 1 : 3. Если даны две модели одной и той же системы, выполняемые на одном и том же компьютере с отношениями 1 : 3 и 1 : 10 соответственно, первый, вероятно, имеет большую степень детальности, чем последний. Однако его меньшая скорость может частично или полностью объясняться низкой эффективностью.

Другим, вероятно более важным, индексом является *разрешающая способность* имитационной модели. Временна́я разрешающая способность может быть определена как кратчайший интервал времени модели между последовательными событиями. Разрешающая способность есть наименьшая идентифицируемая порция информации, представимая в модели, например слово, страница, программа, задание. Другими индексами являются числа состояний и типы событий, рассматриваемые в модели.

Степень детальности не должна быть одинаковой во всей модели. Она часто делается выше в тех частях, о которых предполагается или известно, что их точность в большой степени влияет на точность результатов. Поэтому обычно приходится также рассматривать распределение степени детальности между различными частями модели.

Взаимосвязь между степенью детальности, точностью и ценой модели уже вкратце обсуждались выше. Необходимо только добавить

два важных замечания. Во-первых, устойчивость модели, определяемая как ее способность точно моделировать систему с конфигурациями или рабочими нагрузками, не проверенными экспериментально и даже не предполагавшимися при ее создании, обычно возрастает с ростом детальности. Это будет обсуждаться в разд. 3.3. Во-вторых, необходимая точность зависит от целей работы, например от того, интересуется ли оценщик абсолютными или относительными значениями индексов производительности.

b. *Гибкость*. Построение гибкой модели есть единственный удобный способ решения кратко упомянутой выше проблемы непредсказуемости развития работы. Другой альтернативой является построение модели настолько детальной, что она способна решать все вопросы, которые могут возникнуть. Однако мы уже видели, что этот подход в общем случае должен быть отвергнут. Разумеется, за гибкость тоже приходится платить свою цену временем разработки и временем выполнения, и поэтому она должна вводиться с осторожностью. Во всяком случае, модель должна легко приспосабливаться к изменениям, которых требует процесс калибровки (см. разд. 3.3), а также к изменениям, отражающим модификации рассматриваемой системы и ее рабочей нагрузки в пределах класса работ, для которых и была создана модель.

c. *Язык*. Выбор языка, на котором будет реализована модель, сильно влияет на ее стоимость, так как он сказывается на времени программирования и отладки, времени выполнения, легкости чтения, простоте переноса на другую систему и трудности сопровождения модели. Эта проблема будет обсуждаться в разд. 3.2.5.

d. *Структура модели*. Существуют различные типы моделей для вычислительных систем и их входов. Выбор модели системы влияет на модель рабочей нагрузки и наоборот. Этот выбор частично (а иногда и полностью) зависит от выбора языка. Некоторые из существующих типов моделей описаны в разд. 3.2.2., 3.2.3 и 3.2.4.

e. *Переменные*. Выходные переменные модели соответствуют первичным и вторичным индексам производительности, выбранным для данного исследования. Входные и внутренние переменные выбираются в соответствии со степенью детальности различных частей модели, с ожидаемыми изменениями и доступными данными.

f. *Эксперименты*. Требованиям экспериментов, которые будут проведены на данной модели, следует уделять должное внимание еще при построении модели. В частности, модель должна быть снабжена измерительными средствами, позволяющими собирать необходимые данные во время ее прогонов. Используемые для этой цели измерители обычно можно описать как внутренне управляемые или выборочные программные измерители. Несмотря на то, что модели требуют вычислительных ресурсов от системы, на которой они работают, модель можно построить так, чтобы устранить все искажения в моделируемом времени и моделируемом пространстве. Нужно принять во внимание еще одно соображение, которое касается того, будет ли модель использо-

ваться внутри диалоговой процедуры оценки. В этом случае она должна быть сконструирована в расчете на минимальное время выполнения. Следует предусмотреть необходимые функции обработки данных, которые должны быть встроены в модель. Методы, которые экспериментаторы хотят использовать для планирования экспериментов и, в частности, для определения продолжительности прогонов модели, также оказывают влияние на конструкцию модели (см. разд. 3.4).

3.2.2. Структура модели

В этом разделе мы покажем на примере, как можно ввести структуру простой модели вычислительной системы. Наше обсуждение навеяно работой Мак-Дугалла (1970). Некоторые альтернативные способы организации модели приведены в разд. 3.2.3.

Пример 3.2. Требуется построить простую модель ПРОП-системы пакетной обработки. Главная цель работы состоит в прогнозировании воздействий изменения производительности оборудования на среднюю пропускную способность системы и среднее время прохождения задания.

Из периферийных устройств будем рассматривать устройство чтения перфокарт УЧ и устройство печати УП. Оба устройства управляются процессором P_4, причем считается, что он не создает конфликтов между ними. Другими словами, ситуация такова, как будто УЧ и УП управляются двумя независимыми контроллерами. Задание состоит из ряда *шагов*: компиляции, выполнения, печати колоды перфокарт и т. д. Программа и данные для задания читаются устройством чтения перфокарт УЧ и через входной буфер в памяти М переносятся на диск. Если размер памяти достаточен, информация для первого шага задания просто помещается в память и ждет центрального процессора. Когда наступает ее очередь, она обрабатывается до тех пор, пока не выдает запроса на барабан или на диск. В то время как задание ждет удовлетворения своего запроса, центральный процессор обрабатывает, если они имеются, другие задания, готовые к работе. Когда запрос на барабан удовлетворен, задание вновь начинает ждать центральный процессор в соответствующей очереди. Когда запрос на диск удовлетворен, задание снова поступает в очередь к центральному процессору, если только не завершился текущий шаг этого задания. В последнем случае задание разгружается из памяти и, если есть следующий шаг, становится в очередь к памяти. Заметим, что задания никогда не разгружаются из памяти во время шага задания. Их выходные сообщения накапливаются на диске и в конце концов поступают на печать УП через выделенный выходной буфер в памяти М. Пространство, занятое ими в М, освобождается только по окончании шага задания. Диск используется для программ, их входных данных и выходных сообщений шагов задания. Барабан содержит резидентную часть операционной системы, библиотеку общих программ и подпрограмм, а также личные файлы. Предполагается, что компилятор все время находится в основной памяти. Для удобства изложения, а также поскольку мы чувствуем, что ошибки в результате будут приемлемо малы, мы с самого начала пренебрежем влиянием искажений в памяти и накладных расходов операционной системы на индексы производительности системы. Как можно моделировать эти аспекты поведения системы, будет указано позже.

Изменение производительности компонента оборудования влияет на время, затрачиваемое заданием на работу с этим компонентом. Мы будем рассматривать компоненты оборудования как ресурсы системы. Некоторые ресурсы, например P_1, P_2, P_3, УЧ и УП, нельзя эксплуатировать в режиме разделения времени. Их можно использовать только в одном задании в данный момент времени. Остальные можно разделять между несколькими заданиями. Например, P_4 может совместно использоваться заданием, которое читается с карт, и заданием, результаты которого выводятся на печать. Аналогично, пространство в памяти М может использоваться рядом зада-

ний одновременно. Задание может (или должно) использовать в каждый момент времени более чем один ресурс, например центральный процессор и основную память.

Поскольку устройство ввода перфокарт, печать, барабан и диск могут удовлетворять только один запрос в каждый момент времени, мы можем пренебречь различием между ними и их контроллерами каналов. Таким образом, ресурсы, перечисленные

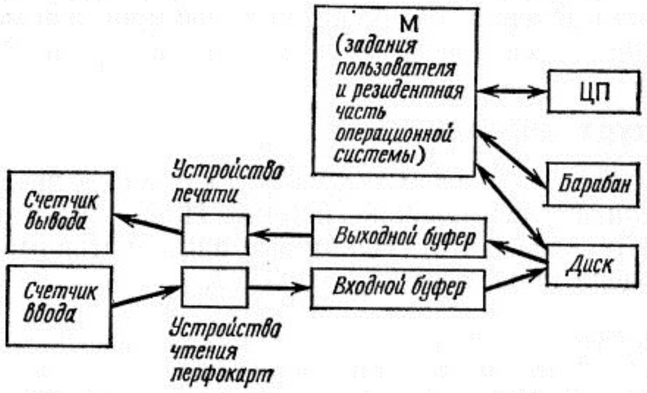


Рис. 3.6. Ресурсы и пути прохождения заданий в ПРОП-системе пакетной обработки из примера 3.2.

выше, можно свести к изображенным на рис. 3.6. Так как входной и выходной буферы можно объединить с УЧ и УП соответственно, наша модель будет иметь дело явно со следующими шестью ресурсами: ЦП, М, БАРАБАН, ДИСК, УЧ и УП.

Для каждой пары «задание — ресурс» мы хотим определить, как долго задание J использует ресурс R. Иными словами, нам нужно определить интервал времени между моментом, когда заданию J *назначен* ресурс R, и моментом, когда J *освобождает* R. Однако прежде, чем ресурс R будет назначен, он должен быть *затребован* заданием J, которое в общем случае должно ждать в *очереди* выполнения своего заказа. Таким образом, мы имеем ситуацию, изображенную на рис. 3.7 (a), где три введенных класса событий (запрос, назначение, освобождение) представлены прямоугольниками, а последовательность действий между ними — стрелками. Надписи на стрелках относятся к деятельности, выполняемой заданием между предыдущим и последующим событиями. Возможное ожидание ресурса изображено символом очереди на соответствующей стрелке.

Модель этой системы может быть организована в соответствии со следующими двумя принципами:

a) строится и непрерывно поддерживается хронологически упорядоченный *список событий*;

b) существует *программа обработки события* для каждого типа события; эта программа вызывается, как только событие этого типа окажется очередным в списке событий.

Таким образом, для каждого ресурса R в этой системе должны существовать три программы: R_ЗАПРОС, R_НАЗНАЧЕНИЕ, R_ОСВОБОЖДЕНИЕ. Эти программы получают имя задания J в качестве аргумента и могут выполнять следующие задачи:

R_ЗАПРОС(J): если R можно немедленно назначить заданию J, тогда создать событие «назначить R заданию J» и изменить состояние R, в противном случае поставить J в очередь к соответствующему R;

R_НАЗНАЧЕНИЕ (J): планировать следующее событие для J (которое не обязательно будет событием «освобождение R»);

R_ОСВОБОЖДЕНИЕ (J): модифицировать состояние R; планировать следующее событие для J; если R можно немедленно назначить заданию K в очереди к R,

то планировать событие «назначить R заданию K» и исключить K из очереди к R; в противном случае ничего не делать.

Заметим, что все программы обработки событий обладают способностью изменять список событий, создавая новые события и планируя их на некоторое будущее время вперед.

Легко видеть, что программа R_НАЗНАЧЕНИЕ (J) может быть объединена с программой R_ЗАПРОС (J), как показано на рис. 3.7 (b), при условии, что основные

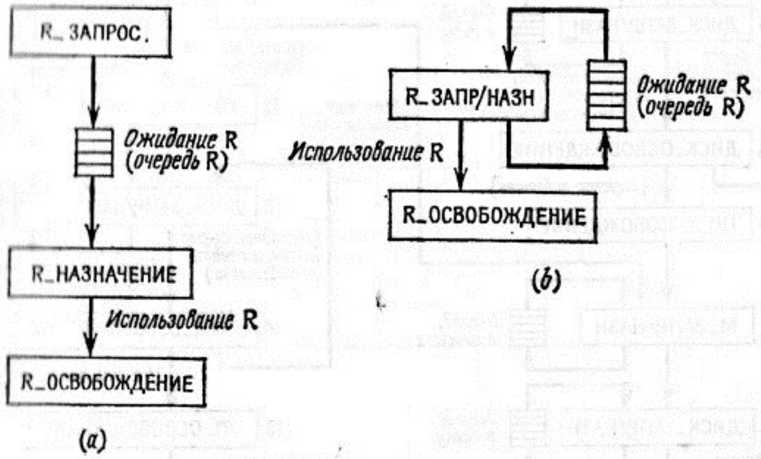


Рис. 3.7. Типы событий в заданиях и действий, относящихся к ресурсу *R*.

операции определенных выше программ модифицированы, например, следующим образом:

R_ЗАПР/НАЗН (J): если R можно немедленно назначить заданию J, то изменить состояние R и планировать следующее событие для J; в противном случае поставить запрос от задания J в очередь к ресурсу R;

R_ОСВОБОЖДЕНИЕ (J): изменить состояние R; планировать следующее событие для J; если очередь к ресурсу R не пуста, планировать событие «запрос/назначение R заданию K» и убрать K из очереди к ресурсу R; в противном случае ничего не делать.

Читателю следует заметить, что в этой схеме постановка в очередь к ресурсу R производится программой R_ЗАПР/НАЗН (J), а исключение — программой R_ОСВОБОЖДЕНИЕ (J). Стратегия планирования и назначения ресурса R осуществляется программой R_ОСВОБОЖДЕНИЕ (J). Когда эта программа выбирает задание для немедленного назначения ресурса R, нужна гарантия от повторного помещения в очередь этого же задания программой R_ОСВОБОЖДЕНИЕ. Такую гарантию можно получить, например, пометив особым образом имя задания (K), которое передается программе R_ЗАПР/НАЗН.

Как определяют следующее событие для задания J описанные выше две программы обработки событий? Чтобы ответить на этот вопрос, мы должны более подробно определить путь, проходимый заданием через нашу систему. Для этой цели необходимо добавить к типам событий, изображенным на рис. 3.7 (b), еще два: события *поступление* и *уход*. Процессы поступления и ухода отличаются от процессов запрос/назначение и освобождение в основном тем, что не существует прямой связи между появлениями и уходами. Если мы рассмотрим всю систему как единый ресурс, то увидим, что этот ресурс немедленно назначается любому поступающему заданию независимо от ухода заданий. (Заметим, что это не так для чисто диалоговой системы; см., например, задачу 3.7.) Таким образом программа ПОСТУПЛЕНИЕ должна планировать появление следующего задания, а программа УХОД оказывается гораздо проще программы ОСВОБОЖДЕНИЕ:

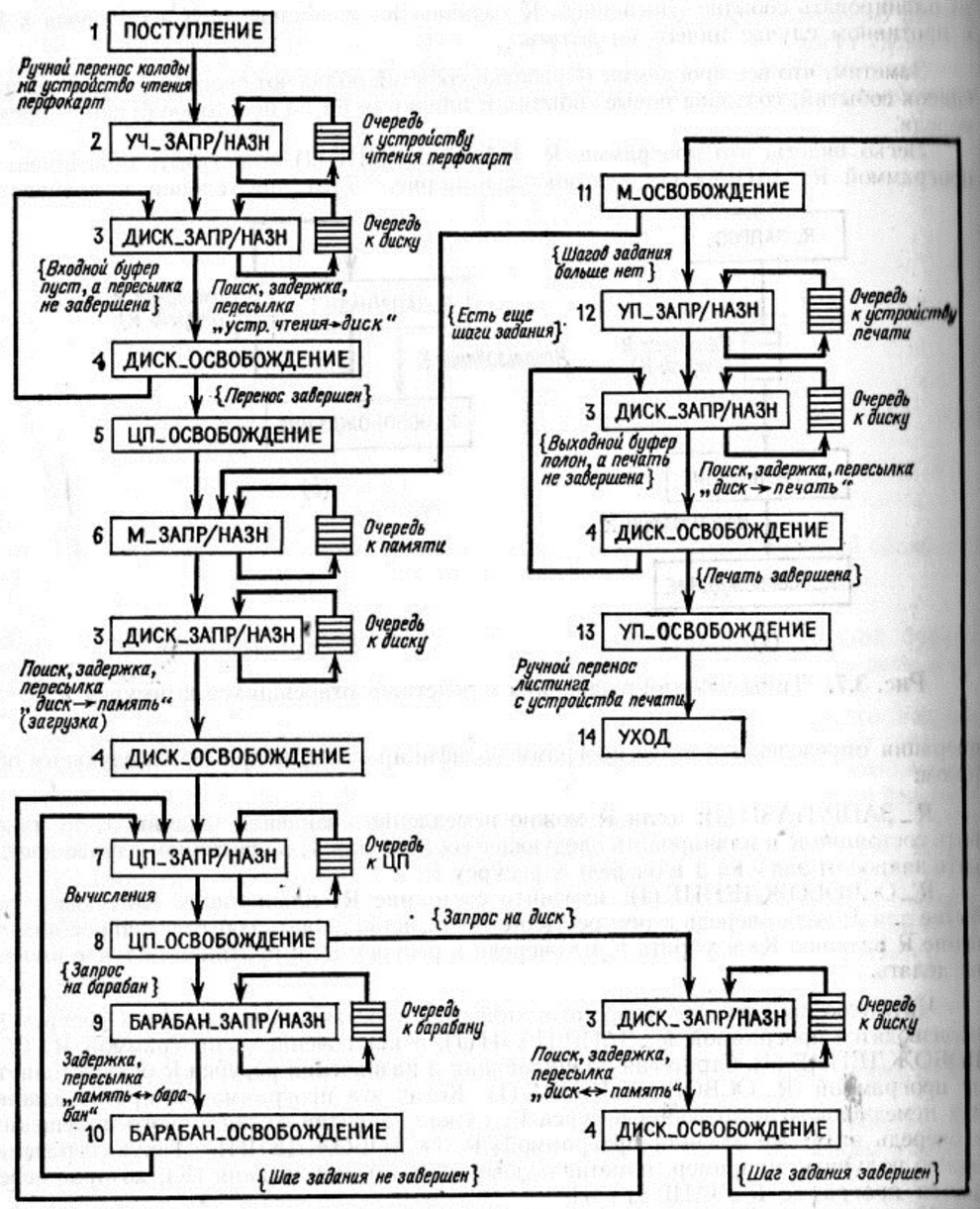


Рис. 3.8. Карта событий ПРОП-системы, рассматриваемой в примере 3.2. Надписи у стрелок описывают действия. Надписи в фигурных скобках содержат условия, при которых выполняется данное действие.

ПОСТУПЛЕНИЕ (J): готовит характеристику задания J; планирует следующее событие для J; планирует прибытие следующего задания, скажем K;

УХОД (J): уничтожает характеристику J.

Мы теперь можем построить то, что мы назовем *картой событий* нашей системы. Эта карта, показанная на рис. 3.8, отражает путь задания через систему, которую мы хотим моделировать. Некоторые предположения относительно поведения системы

были упомянуты выше, но легко могут быть получены из карты, которую читатель должен внимательно изучить во всех деталях.

На карте событий, изображенной на рис. 3.8, имеется 20 прямоугольников, соответствующих событиям, 14 из них (по два на каждый из шести ресурсов рис. 3.6 плюс события «поступление» и «уход») соответствуют различным типам событий. Модель состоит из 14 программ, обрабатывающих эти типы событий. Программы можно

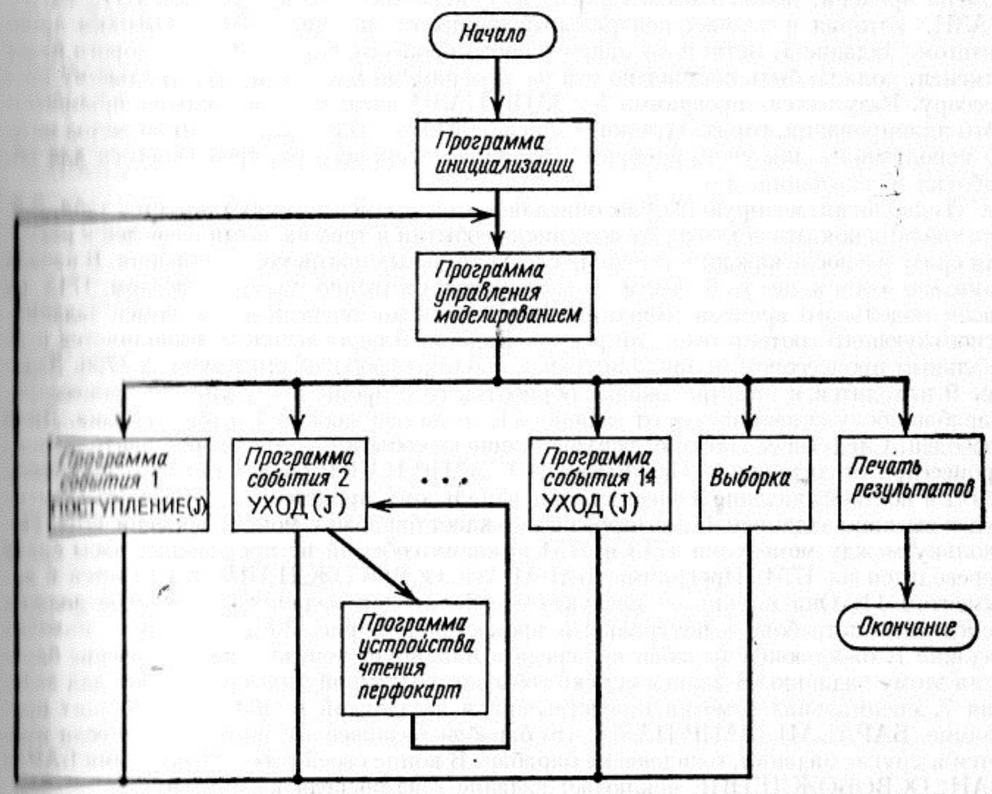


Рис. 3.9. Блок-схема моделирующей программы, описанной в примере 3.2.

организовать согласно блок-схеме рис. 3.9. *Программа инициализации* просто строит структуры данных модели и приводит их в начальное состояние. *Программа управления моделью* (ПУМ) выполняет две основных задачи, а именно:

a) выбрать следующее событие и вызвать соответствующую программу обработки события;

b) перевести часы модели, измеряющие время модели.

Обе задачи могут быть выполнены путем проверки верхней записи из списка событий. Каждая запись в списке содержит тип события, который используется для вызова соответствующей программы обработки события (например, ЦП_ОСВОБОЖДЕНИЕ), имя задания, которое нужно передать как аргумент программе обработки события, а также время события, которое должно быть загружено в часы модели программой управления моделью. Момент возникновения события определяется программой обработки события, которая его генерирует. Например, программа ЦП_ЗАПР/НАЗН (J) в некоторых случаях вычисляет момент времени модели, когда будет прервано выполнение J из-за запроса на диск или барабан (напомним, что окончание шага задания вызывает запрос диска). Это вычисление основано на текущем значении времени модели и на длительности следующего вычисленного интервала для задания J, который можно получить из описания задания (см. разд. 3.2.4).

Что происходит в том случае, если используется *приоритетное* планирование центрального процессора, т. е. если заданию, имеющему более высокий приоритет, чем выполняемое в данный момент задание J, будет разрешено немедленно получить управление центрального процессора, как только оно будет включено в очередь к центральному процессору, тем самым вызвав непредвиденное прерывание J? Событие «освобождение центрального процессора» для J, запланированное на определенный момент времени, должно быть удалено из списка событий программой ЦП_ЗАПР/НАЗН, которая назначает центральный процессор заданию с более высоким приоритетом. Задание J, остаток от запрошенного процессорного времени которого будет изменен, должно быть поставлено той же программой в очередь к центральному процессору. Разумеется, программа R_ЗАПР/НАЗН, необходимая в случае приоритетного планирования, гораздо сложнее описанной выше. Заметим, что тот же метод можно использовать для учета времени центрального процессора, требующегося для обработки прерываний.

Технология, которую мы уже описали в этом примере, иллюстрируется табл. 3.3. Эта таблица показывает содержимое списка событий и трех из шести очередей к ресурсам сразу же после каждого из семи последовательных шагов моделирования. В начале показано состояние этой части модели непосредственно перед моментом 1713 по часам модельного времени. Верхний элемент каждой очереди — это номер задания, использующего соответствующий ресурс. Задание 5 первоначально выполняется центральным процессором, и запланировано, что оно освободит его в момент 1756. Задание 9 находится в памяти, ожидая обработки со стороны центрального процессора. Барабан обслуживает запрос от задания 11, тогда как задание 7 ждет барабана. Диск свободен. Следующее событие, верхнее в списке событий,— это запрос центрального процессора от задания 3. Программа ЦП_ЗАПР/НАЗН обнаружит, что процессор занят и поставит задание 3 в очередь к центральному процессору. Следующее событие будет вызвано заданием 11, которое освобождает барабан в момент времени 1754. Поскольку между моментами 1713 и 1754 никаких событий не происходит, часы сразу переводятся на 1754. Программа БАРАБАН_ОСВОБОЖДЕНИЕ вызывается с аргументом 11. Она планирует следующее событие для задания 11, которое должно немедленно затребовать центральный процессор (см. рис. 3.8). Она также находит задание 7, ожидающее барабан в очереди и планирует немедленное назначение барабана этому заданию. В записи списка событий, созданной этой программой для задания 7, специальная отметка (представленная звездочкой в табл. 3.3) сообщит программе БАРАБАН_ЗАПР/НАЗН, что барабан назначен заданию 7, даже если имеются и другие задания, ожидающие барабан. В конце своей работы программа БАРАБАН_ОСВОБОЖДЕНИЕ исключает задание 7 из очереди к барабану.

Центральный процессор освобождается заданием 5 в момент 1756. Программа ЦП_ОСВОБОЖДЕНИЕ назначает центральный процессор заданию 9 и планирует немедленный запрос диска для задания 5. Поскольку диск свободен, он сразу же назначается заданию 5, тогда как центральный процессор начинает обработку задания 9.

Время, необходимое для планирования следующего события, может быть вычислено с различными степенями детальности и, следовательно, с разной точностью. Например, может планироваться, что запрос на барабан или диск возникает сразу же за событием «освобождение центрального процессора», без вмешательства центрального процессора, как было сделано в табл. 3.3. В ином случае можно принять во внимание системные накладные расходы (поиск в справочнике, вычисление адреса на диске или барабане, операции учета) с той или иной степенью детальности. Другим примером является модель диска, используемая программой ДИСК_ЗАПР/НАЗН для вычисления общего времени $t_{дс}$, в течение которого диск обслуживает один запрос. Это может быть очень простая и грубая аналитическая модель следующего типа:

$$t_{дс} = \overline{t}_{дс}, \tag{3.2}$$

$$t_{дс} = \alpha, \tag{3.3}$$

$$t_{дс} = \overline{t}_{адс} + t_{сдс} r_{дс}, \tag{3.4}$$

$$t_{дс} = \overline{t}_{поиск} + \beta R_{дс} + t_{сдс} r_{дс}. \tag{3.5}$$

Таблица 3.3

Последовательное содержимое списка событий и очередей к ресурсам во время прогона модели (см. пример 3.2)

Список событий			Очередь к ЦП	Очередь к барабану	Очередь к диску
Тип события	Номер задания	Время события	Номер задания	Номер задания	Номер задания
ЦП_ЗАПР/НАЗН	3	1713	5	11	0
			9	7	0
БАРАБАН_ОСВОБОЖДЕНИЕ	11	1754			
ЦП_ОСВОБОЖДЕНИЕ	5	1756			
БАРАБАН_ОСВОБОЖДЕНИЕ	11	1754	5	11	0
			9	7	0
ЦП_ОСВОБОЖДЕНИЕ	5	1756			
			3		
ЦП_ЗАПР/НАЗН	11	1754	5	0	0
			9	0	0
БАРАБАН_ЗАПР/НАЗН	7 *	1754			
ЦП_ОСВОБОЖДЕНИЕ	5	1756	3		
БАРАБАН_ЗАПР/НАЗН	7 *	1754	5	0	0
			9	0	0
ЦП_ОСВОБОЖДЕНИЕ	5	1756			
			3		
			11		
ЦП_ОСВОБОЖДЕНИЕ	5	1756	5	7	0
			9	0	0
БАРАБАН_ОСВОБОЖДЕНИЕ	7	1792	3		
			11		
ДИСК_ЗАПР/НАЗН	5	1756	0	7	0
			3	0	0
ЦП_ЗАПР/НАЗН	9 *	1756			
БАРАБАН_ОСВОБОЖДЕНИЕ	7	1792	11		
ЦП_ЗАПР/НАЗН	9 *	1756	0	7	5
			3	0	0
БАРАБАН_ОСВОБОЖДЕНИЕ	7	1792	11		
ДИСК_ОСВОБОЖДЕНИЕ	5	1815			
			9	7	5
			3	0	0
ЦП_ОСВОБОЖДЕНИЕ	9	1783			
			11		
БАРАБАН_ОСВОБОЖДЕНИЕ	7	1792			
ДИСК_ОСВОБОЖДЕНИЕ	5	1815			

* Заданию должен быть немедленно назначен ресурс.

В уравнениях (3.2)—(3.5) α — случайная переменная с заданным распределением, β — случайная переменная, равномерно распределенная в интервале от 0 до 1, $t_{сдс}$ — (постоянное) время пересылки слова для диска, $r_{дс}$ — длина затребованной

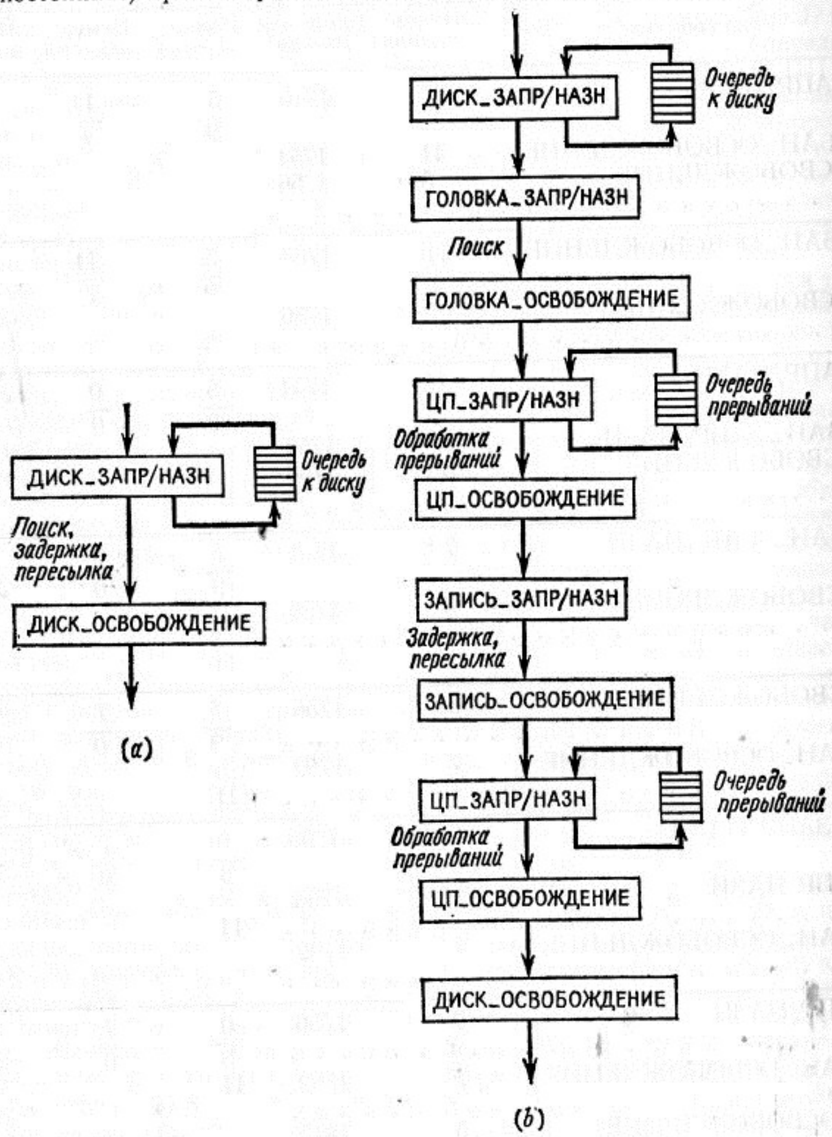


Рис. 3.10. Две карты событий для запроса диска: (a) — карта, использованная на рис. 3.8; (b) — карта, соответствующая более детальной модели диска.

записи в словах, $R_{дс}$ — (постоянное) время оборота диска, а средние [1] считаем константами, полученными из измерений или из спецификаций изготовителя.

Однако может использоваться и более замысловатая аналитическая модель диска [см., например, гл. 5 книги Коффмана и Деннинга (1973)] или даже более точная имитационная модель, которая сохраняет опознавательную дорожку и расположение записей наряду с текущими положениями диска и головок. Карта событий одной из

[1] Здесь $\bar{t}_{ддс}$ — среднее время доступа к диску.— *Прим. ред.*

ности возможных моделей последнего типа приведена на рис. 3.10(b). На рис. 3.10(a) показана соответствующая часть общей карты событий из рис. 3.8. Мы предположили, что диск управляется специализированным контроллером с фиксированной программой. Когда диск назначается некоторому заданию, центральный процессор посылает контроллеру команду поиска. Время поиска вычисляется программой обработки события ГОЛОВКА_ЗАПР/НАЗН по характеристике поиска, которая является кривой, связывающей $t_{поиск}$ с числом цилиндров, которые предстоит пересечь. Когда поиск завершен, работа центрального процессора прерывается. Поскольку может оказаться, что прерывания от контроллера диска маскированы или обладают приоритетом более низким, чем у выполняемой в данный момент программы, мы должны ввести очередь прерываний. Когда прерывание обработано, центральный процессор выдает команду пересылки, которая специфицирует направление пересылки, количество пересылаемой информации, а также адреса на диске и в памяти, между которыми должна произойти пересылка. Прежде чем мы сможем пересылать, должно пройти время подготовки. Это время вычисляется программой ЗАПИСЬ_ЗАПР/НАЗН, исходя из мгновенного углового положения диска и адреса на диске для пересылки. Та же программа вычисляет время пересылки и планирует вызов программы ЗАПИСЬ_ОСВОБОЖДЕНИЕ по прошествии интервала времени, равного сумме времен подготовки и пересылки. Наконец, программа ЗАПИСЬ_ОСВОБОЖДЕНИЕ планирует немедленное прерывание центрального процессора. Когда прерывание обработано, программа ЦП_ОСВОБОЖДЕНИЕ планирует событие «освобождение диска», которое завершает обслуживание запроса на диск.

Широкий спектр точностей, стоимостей программирования и выполнения характеризует возможные модели каждого компонента системы. Кроме того, данные, предоставляемые пользователем для каждого задания и накапливаемые в структурах данных имитационной модели, существенно меняются от модели к модели. Это очевидно, если сравнить модель в (3.2) с моделью рис. 3.10(b). Выбор моделей для различных комбинаций ресурсов в идеальном случае диктовался бы критерием минимальной сложности при заданной точности. Этому критерию, однако, совсем не легко следовать. Обычно отсутствие подходящей входной информации действует в ту же сторону, что и критерий минимальной сложности, т. е. оно удерживает сложность моделей на относительно низком уровне. И в самом деле, бессмысленно конструировать сложные модели, требующие знания рабочих характеристик, которые невозможно получить из опыта и которые неизбежно придется изобретать.

Наша имитационная модель должна использоваться для анализа производительности различных конфигураций ПРОП-системы, отличающихся друг от друга производительностью компонентов оборудования. Следовательно, параметры, представляющие соответствующие индексы производительности [например, $\bar{t}_{поиск}$, $R_{дс}$ и $t_{сдс}$ в дисковой модели (3.5)], должны содержаться среди аргументов программ, реализующих модели этих компонентов, и среди входных переменных нашей модели. Модель, выбранная для диска, может быть встроена в программу ДИСК_ЗАПР/НАЗН, которая является единственной программой, использующей его. Однако иметь отдельную программу для диска, вызываемую программой ДИСК_ЗАПР/НАЗН, обычно гораздо удобнее, особенно если специалист ожидает, что модель диска потребует изменений (для увеличения или уменьшения степени ее детальности) в течение периода калибровки или на этапе эксплуатации модели.

На карте событий рис. 3.8 имеется несколько событий, за каждым из которых следует по два события. Эти последующие события являются взаимно исключающими и должны быть выбраны подходящей программой обработки события в соответствии с характеристиками обрабатываемого задания. Заметим также, что некоторые программы обработки событий могут приводить к различным последующим событиям. Например, программа ДИСК_ОСВОБОЖДЕНИЕ после активации будет планировать одно из следующих событий для задания J: *затребовать/назначить диск, освободить устройство чтения перфокарт, затребовать/назначить центральный процессор, освободить память, освободить устройство печати.* Событие *затребовать/назначить центральный процессор* планируется, если задание J только что было загружено в основную память или если запрос на диск, выданный заданием J, только что был удовлетворен, а вычисления для текущего шага задания еще не завершены. Таким

образом, информация о характеристиках и состоянии каждого задания в системе, необходимая для принятия решений, должна содержаться в модели и непрерывно обновляться программами обработки событий.

Средства, необходимые для измерения значений выходных переменных во время моделирования, легко могут быть смоделированы сами. Например, чтобы получить среднюю пропускную способность $\bar{T}$, определяемую как среднее число заданий, обработанных в единицу времени, мы можем просто подсчитать число завершенных заданий, введя программометр в программу УХОД. Игнорируя проблемы начального смещения и хвостового эффекта, которые будут рассматриваться в разд. 3.4.2, найдем, что средняя пропускная способность на протяжении всего прогона модели определяется как отношение последнего отсчета программометра к конечному значению времени модели. Разумеется, мы предполагаем, что и программометр и часы модели были в начале прогона установлены на нуль.

Часто в качестве критерия остановки экспериментатор использует либо общее время, либо общее число заданий, подлежащих обработке. В этом случае нужно измерить только вторую из этих величин.

Значение $\bar{T}$ как функция времени, т. е. пропускная способность, усредненная за время от начала прогона до текущего момента модели, может быть получено, прослеживая и запоминая моменты ухода заданий $t_{d1}, t_{d2}, \ldots, t_{dn}$. Средняя пропускная способность в момент t, где $t_{d,i} \leqslant t < t_{d,i+1}$, равна i/t. Заметим, однако, что эта величина может быть вычислена в любой момент во время моделирования, если просто взять отношение показаний программометра к показаниям часов: при этом нет нужды прослеживать и запоминать все моменты ухода заданий.

Среднее значение времени прохождения $\bar{t}$ получается, согласно определению среднего, из значений времени прохождения отдельных заданий. Время прохождения задания вычисляется программой УХОД как разность между временем его ухода (окончания) и временем поступления (начала). Вместо запоминания всех значений времен прохождения программа УХОД может хранить текущую оценку среднего $\bar{t}$, которая в каждый момент является значением $\bar{t}$, накопленным к данному моменту. Если мы обозначим через $\bar{t}_i$ время прохождения, среднее по первым i заданиям, обработанным моделью, а через t_j — время прохождения j-го обработанного задания, то получим

$$\bar{t}_i = \frac{1}{i} \sum_{j=1}^{i} t_j. \tag{3.6}$$

Записывая (3.6) для $\bar{t}_{i+1}$ и подставляя туда (3.6), получим

$$\bar{t}_{i+1} = \frac{i}{i+1} \bar{t}_i + \frac{1}{i+1} t_{i+1}. \tag{3.7}$$

Таким образом, чтобы вычислить $\bar{t}_{i+1}$, нам не нужны значения $t_1, t_2, \ldots, t_i$. Достаточно знать $\bar{t}_i$ (предыдущее значение этого индекса производительности), i и t_{i+1} (время прохождения задания, в данный момент времени покидающего систему).

Кроме того, дисперсия σ_i^2 последовательных наблюдений относительно их текущего среднего $\bar{t}_i$ может храниться и модифицироваться в ходе моделирования без сохранения всех отдельных времен прохождения. Используя определение дисперсии и предполагая σ_i^2 известным, получим с помощью нескольких алгебраических действий

$$\sigma_{i+1}^2 = \frac{i}{i+1} \sigma_i^2 + \frac{i}{(i+1)^2} (t_{i+1} - \bar{t}_i)^2. \tag{3.8}$$

Так как $\sigma_1^2 = 0$, то (3.8) позволяет рекурсивно вычислять все последовательные значения дисперсии. Это вычисление требует знания только i, σ_i^2, t_{i+1} и $\bar{t}_i$.

Иногда для уменьшения количества данных, которые приходится накапливать, и затрат времени на сбор и обработку данных предпочтительно моделировать выбороч-

ный измеритель вместо внутренне управляемого, который записывает все интересующие нас события. Выборка может быть выполнена посредством добавления к модели своей программы обработки события (программа ВЫБОРКА на рис. 3.9), которая, будучи активирована, собирает все необходимые данные из структур данных модели. Подобно программе ПОСТУПЛЕНИЕ, программа ВЫБОРКА планирует свой следующий вызов перед возвратом в программу управления моделью. Некоторые проблемы, возникающие при измерении и интерпретации данных моделирования, обсуждаются в разд. 2.6.4, 2.7, 3.4.2 и 3.5. ■

3.2.3. Другие организации моделей

Подход, принятый в разд. 3.2.2 для построения модели пакетной СМО-системы, основывался на понятиях *списка событий и программы обработки события*. Программа управления моделированием использует список событий для перевода часов времени модели на момент следующего события. Метод перевода часов через интервалы, не содержащие событий, известен как *метод очередных событий*. При этом интервалы между последовательными событиями могут оказаться непостоянными. В этом типе моделей событию, естественно, придается особое значение. Каждому типу событий в модели соответствует подпрограмма, которая вызывается, когда происходит событие этого типа, и которая в свою очередь порождает одно или несколько событий и планирует их появление. Тогда говорят, что модель, подобно построенной в разд. 3.2.2, основана на *подходе планирования событий*. Читатель уже наверняка заметил, как изящно воспроизводит такая простая конструкция, как список событий, поведение конкурирующих процессов.

Модель с существенно отличающейся структурой получается, если ее создатели сосредоточивают внимание на видах деятельности или на процессах (т. е. на последовательностях событий, проходимых каждым заданием), а не на отдельных событиях.

Если особое внимание уделяется видам деятельности, события не планируются, а время модели получает постоянные приращения. На каждом шаге для каждого из текущих видов деятельности программа, управляющая моделированием, проверяет, выполнены ли условия начала или окончания этого вида деятельности. Этот метод увеличения показаний часов называется *методом постоянных шагов*, а сам подход известен как *последовательный просмотр (сканирование) видов деятельности*. Простейшим способом структурирования модели, основанной на этом подходе, является вызов программой управления моделью *программ видов деятельности*, как только проверка условий покажет, что можно начать или завершить соответствующий вид деятельности в данный момент времени. Программа вида деятельности, кроме всего прочего, определяет условия, при которых данный вид деятельности должен быть прекращен или возобновлен. Это можно рассматривать как планирование условных событий, которые произойдут только при удовлетворении определенных условий. Например, свободный ресурс, очередь к которому пуста, может снова оказаться занятым, когда в не-

который непредсказуемый момент времени в будущем на него появится новый запрос. В терминах языка программирования вместо безусловного оператора **wait** ⟨*время*⟩, как в модели с планированием событий, этот подход использует условные операторы, подобные **wait until** ⟨*условия*⟩.

При моделировании вычислительных систем последовательный просмотр видов деятельности с постоянным шагом по времени обычно менее удобен, чем планирование событий, поскольку события имеют тенденцию распределяться во времени очень неравномерно. Кроме того, их точная хронологическая последовательность, вообще говоря, слишком важна, чтобы ее игнорировать при выборе шагов относительно большой длительности по времени. Добавим также, что существует много видов деятельности, продолжительности которых могут быть точно вычислены в момент их начала.

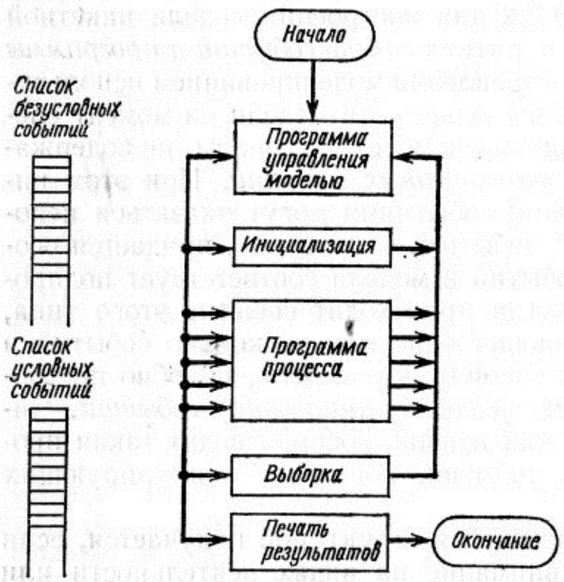


Рис. 3.11. Возможная структура моделирующей программы, основанная на подходе с взаимодействием процессов.

Моделирование можно ускорить, применяя метод переменных приращений времени, т. е. комбинируя планирование событий со сканированием видов деятельности. Возможную организацию для такой комбинации можно представить следующим образом. Программа управления моделью делает значение времени модели равным времени очередного запланированного (безусловного) события и вызовет выполнение всех программ событий, которые должны были произойти к этому моменту времени. Затем она просматривает список условных событий и выполняет все события, условия которых удовлетворены. Этот просмотр должен повторяться до тех пор, пока не останется условных событий, которые могли бы быть выполнены. Затем часы можно перевести на момент следующего запланированного события. Если модель ориентирована на виды деятельности, это можно назвать подходом, *использующим последовательный просмотр видов деятельности с переменным шагом по времени*. Заинтересованный читатель найдет подробное изложение этого вопроса в разд. 3.3.3 книги Фишмана (1973).

Более распространенный принцип построения модели, использующий сочетание условных и безусловных событий, принят в *подходе с взаимодействием процессов*. Здесь внимание сосредоточено на истории прохождения задания через систему и на взаимодействиях парал-

лельных, частично перекрывающихся и взаимодействующих историй прохождения заданий, оказавшихся одновременно в системе.

Один из способов структурирования модели, основанный на подходе с взаимодействием процессов, иллюстрируется рис. 3.11. Программа процесса описывает путь (пути,) проходимый заданием в системе. Это описание подобно описанию, данному на рис. 3.8 для примера 3.2. Если следующее событие для задания, история прохождения которого моделируется, в некоторый момент времени можно планировать безусловно, программа процесса делает это, добавляя соответствующую запись к списку безусловных событий. Если это невозможно, например

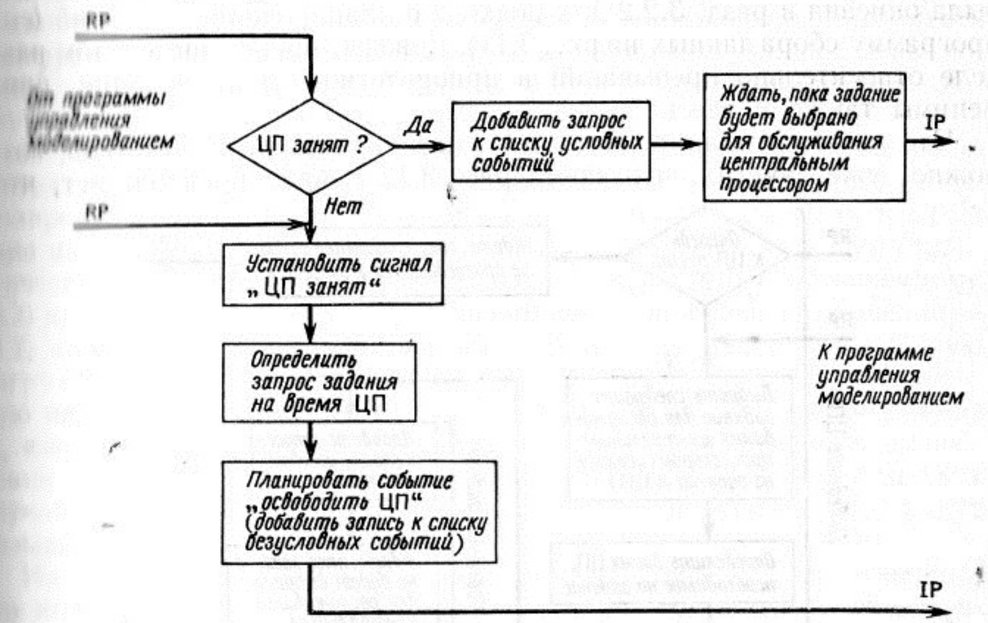


Рис. 3.12. Блок-схема части программы процесса (см. рис. 3.11), представляющая запрос ЦП заданием и назначение его заданию (версия 1 подхода с взаимодействием процессов). Здесь RP — точка реактивации; IP — точка взаимодействия.

из-за взаимодействия с другими заданиями, которые занимают запрошенный ресурс, запрос добавляется к списку условных событий. Затем управление возвращается программе управления моделью, которая осуществит возврат в программу процесса через точку входа, непосредственно следующую за оператором **wait until**, когда задание будет выбрано для обслуживания этим ресурсом. Таким образом, программа процесса имеет ряд точек входа (или точек *реактивации*) и выхода (точек *взаимодействия*). Когда происходит возврат управления к программе управления моделью, эта программа просматривает списки условных и безусловных событий, как описано выше, для определения своих последующих действий.

На рис. 3.12 представлена блок-схема части программы процесса для модели системы, рассмотренной в примере 3.2. Изображенная часть относится к обработке запросов на центральный процессор и наз-

значению центрального процессора некоторому заданию. Читателю следует обратить внимание на то, что производительность центрального процессора (например, его скорость) учитывается в блоке, названном «определить запрос задания на время центрального процессора». В модели, построенной для исследования, описанного в начале примера 3.2, быстродействие центрального процессора нужно включить в число входных параметров. Этот блок следует модифицировать или заменить, если по некоторым причинам придется изменить модель центрального процессора. Периодическое получение данных в модели этого типа можно осуществить с помощью такой же технологии, какая была описана в разд. 3.2.2 для подхода с планированием событий (см. программу сбора данных на рис. 3.11). Доводы, приведенные в том разделе относительно прерываний и приоритетного планирования, применимы также и здесь.

Что касается планирования и назначения ресурсов, читатель, возможно, уже заметил, что схема рис. 3.12 неявно предполагает, что

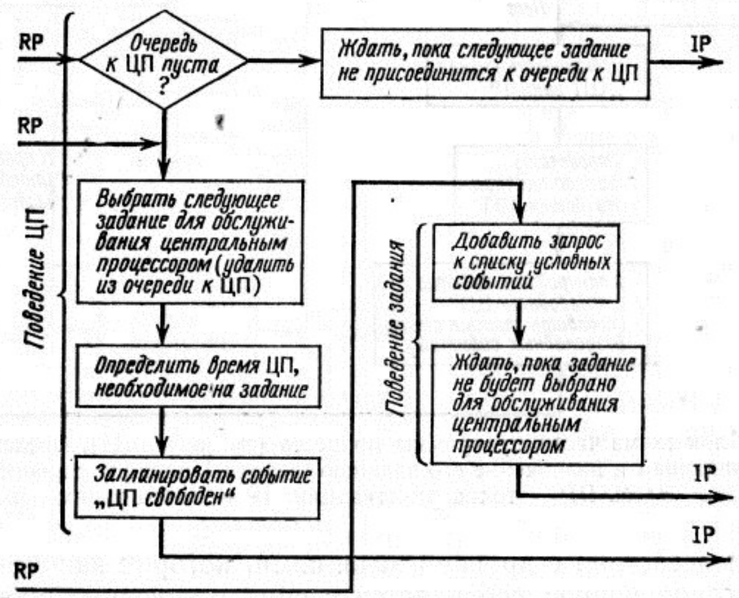


Рис. 3.13. Блок-схема программы процесса (см. рис. 3.11), представляющая запрос ЦП заданием и его назначение заданию (версия 2 подхода с взаимодействием процессов). Здесь RP — точка реактивации; IP — точка взаимодействия.

эти задачи выполняются программой управления моделью, а не программой процесса. Было бы неестественным, хотя и возможным, модифицировать эту схему так, чтобы эти задачи выполнялись программой процесса, которая воспроизводит историю заданий. Действительно, ресурсы рассматриваются как пассивные объекты и данная версия подхода с взаимодействием процессов (мы ее назовем *версия 1*) не обеспечивает явного моделирования поведения ресурсов.

Другая версия *(версия 2)* этого же подхода моделирует поведение ресурсов наряду с поведением заданий. Пример участка, относящегося

к запросу/назначению центрального процессора, в этой версии нашей модели показан на рис. 3.13. Сравнение этого рисунка с рис. 3.12 поможет читателю понять концептуальное сходство и различия между двумя версиями. В частности, легко видеть, что стратегия планирования реализуется в версии 2 блоком, названным «выбрать следующее задание для обслуживания центральным процессором».

3.2.4. Моделирование рабочей нагрузки

Модель рабочей нагрузки, которая будет управлять моделью вычислительной системы во время ее прогонов, должна формулироваться одновременно с моделью системы. В этом разделе мы опишем типы моделей, которые можно использовать для моделирования рабочей нагрузки установки.

Первое полезное различие, относящееся также и к моделям систем, это различие *детерминированных* и *стохастических* моделей. В детерминированной модели все переменные детерминированы. Если хотя бы одна из переменных случайна, то говорят, что модель является стохастической. Например, модели диска, выраженные соотношениями (3.3) или (3.5), стохастические, тогда как представленные сотношениями (3.2) или (3.4) — детерминированные. Однако, характеризуя рабочую нагрузку, можно рассматривать длину записи $r_{дс}$ в (3.4) как случайную переменную. Это означает, что мы можем задать распределение $r_{дс}$ вместо длины каждой отдельной записи, включенной в моделирование. В этом случае, поскольку модель рабочей нагрузки содержит случайную переменную $r_{дс}$, эта модель и модель всей системы будут называться стохастическими.

Из уже рассмотренных простых примеров должно быть очевидно, что количество информации, необходимой для адекватного представления многих параметров рабочей нагрузки, может быть существенно сокращено только путем применения стохастической модели рабочей нагрузки. Это не означает, что детерминированные модели рабочей нагрузки не находят применения. Наоборот, два типа таких моделей использовались и используются успешно: модели, содержащие *трассировочную запись* входных событий, обычно во время сеанса измерения в модели, описывающие рабочую нагрузку в форме *программы*, возможно с использованием специализированного языка. Таким образом, в зависимости от типа модели рабочей нагрузки мы имеем три типа моделей вычислительных систем:

a) *модели, управляемые трассировочной записью,*
b) *модели, управляемые программно,*
c) *модели, управляемые распределениями.*

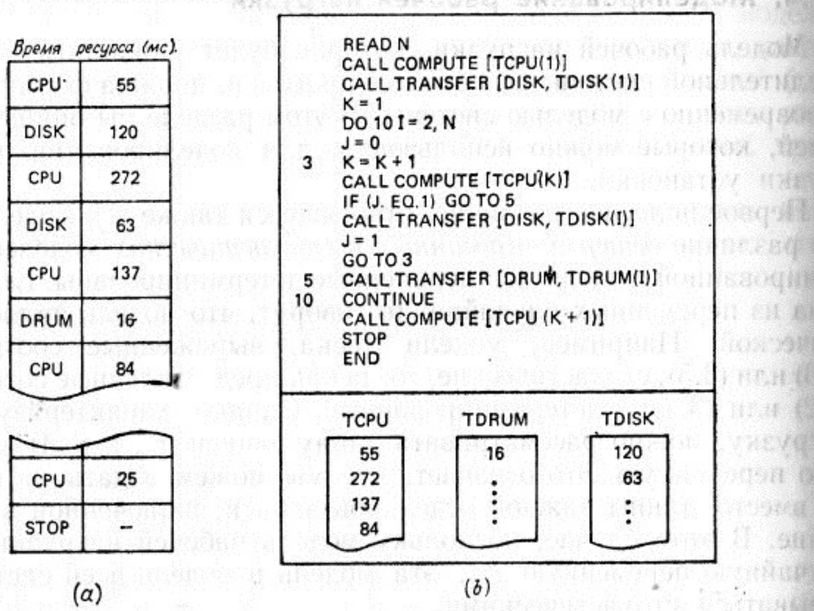
В категорию (c) мы включаем все модели систем, управляемые стохастическими моделями рабочей нагрузки.

Читателю следует иметь в виду, что модель не обязательно должна принадлежать в точности одной из этих категорий. Возможны любые

комбинации этих трех методов моделирования рабочей нагрузки, и некоторые из них действительно применяются в моделях систем.

Чтобы проиллюстрировать введенные концепции, мы сейчас подробно опишем их применение к моделям, о которых шла речь в примерах 3.1 и 3.2.

Пример 3.3. Рабочая нагрузка для ПРОП-системы из примера 1 состоит из единственной программы. Параметрами программы, существенными для нашего моделирования, являются те, которые определяют переходы между состояниями, изображенные на рис. 3.3 (b). Эти переходы совпадают с событиями, перечисленными в табл. 3.1. Модель, управляемая трассировочной записью, может быть построена так, что она будет принимать трассировочную запись, изображенную на рис. 3.14 (a). Каждый элемент этой трассировочной записи содержит указатель ресурса и время, в течение которого программа использовала этот ресурс.



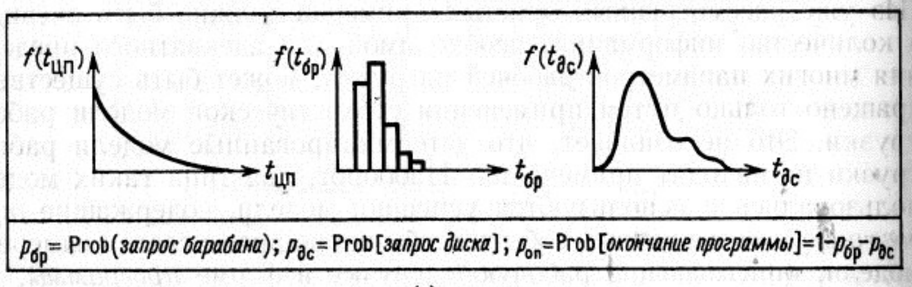


Рис. 3.14. Модели программ для примера 3.3: (a) — модель для моделирующей программы, управляемой трассировочной записью; (b) — модель для программно-управляемой моделирующей программы; (c) — модель для моделирующей программы, управляемой распределением.

На рис. 3.14 (b) представлена модель программы, которая может быть помещена в программно управляемую модель системы. Эта модель дана на понятном без пояс-

ний языке, подобном ФОРТРАНУ и ориентированном специально на такое применение. Программа рис. 3.14 (b) генерирует трассировочную запись рис. 3.14 (a) по ходу моделирования системы. Подпрограммы COMPUTE и TRANSFER можно считать содержащими вызовы модели системы, которая в данном случае крайне проста.

Легко видеть, что эта модель более компактна, но накладывает больше ограничений, чем модель рис. 3.14 (a). Обе модели характеризуют программу длительностями интервалов пребывания в каждом из трех состояний без прерываний S_1, S_2 и S_3 и полным порядком, в котором затребованы ресурсы, соответствующие этим состояниям.

Если программа вызывает очень большое количество переходов между состояниями или если мы заинтересованы скорее в изучении поведения класса программ, чем в определении времен выполнения отдельной программы, мы можем характеризовать программу распределениями времен пребывания в каждом состоянии и вероятностями переходов после того, как система покинет состояние S_1. Программная модель этого типа схематически представлена на рис. 3.14 (c), где $f(x)$ означает плотность

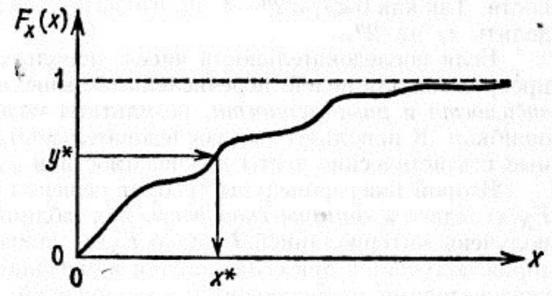


Рис. 3.15. Процедура генерации значений случайной переменной $X \geqslant 0$, имеющей заданную функцию распределения $F_X(x)$. Обратите внимание на то, что Y должна быть случайной переменной с равномерным распределением от 0 до 1.

распределения случайной переменной x. Каждый раз когда моделирующей программе необходимо узнать продолжительность пребывания в состоянии, в котором она только что оказалась, она делает случайное испытание по соответствующему распределению.

Как можно генерировать значения случайной переменной, имеющей такую же функцию распределения вероятности $F_X(x)$, что и заданная случайная переменная X? Пусть диаграмма на рис. 3.15 представляет $F_X(x)$. Предположим для простоты, но без потери общности, что X может принимать только неотрицательные значения. Это верно, например, для времен пребывания в состояниях. По определению $F_X(x)$ — это вероятность того, что значение случайной переменной X меньше или равно данному x. Поэтому $F_X(x)$ является монотонно неубывающей функцией, изменяющейся от 0 до 1. Обычная процедура получения значения $F_X(x)$ состоит из следующих шагов (см. рис. 3.15):

a) генерируется случайное число выборкой случайной переменной Y, равномерно распределенной на интервале (0,1); пусть y^* — ее значение;

b) определяется x^*, такое значение x, что $F_X(x^*) = y^*$.

Каково распределение случайной переменной X^*, значения которой получены с помощью описанной выше процедуры? Мы имеем

$$\text{Prob}\,(X^* \leqslant x) = \text{Prob}\,[Y \leqslant y = F_X(x)]; \tag{3.9}$$

следовательно,

$$F_{X^*}(x) = F_Y(F_X(x)). \tag{3.10}$$

Дифференцируя обе части (3.10) по x, мы получим следующее уравнение для функций плотности вероятности:

$$f_{X^*}(x) = f_Y(y) \cdot f_X(x). \tag{3.11}$$

Поскольку Y распределена равномерно на интервале (0,1), в соответствующем интервале значений x мы имеем $f_Y(y)=1$. Таким образом, (3.11) принимает вид

$$f_{X^*}(x)=f_X(x). \tag{3.12}$$

Это показывает, что распределение X^* совпадает с распределением X.

Первый шаг процедуры генерации последовательности независимых и одинаково распределенных случайных переменных требует наличия механизма, производящего равномерно распределенные случайные числа. Возможны различные алгоритмы генерации псевдослучайных чисел. Программа в табл. 2.8, например, является *линейным конгруэнтным генератором*, основанным на рекурсивной формуле:

$$r_i=(mr_{i-1}+a) \bmod 2^n \qquad (i=1, 2, \ldots). \tag{3.13}$$

В (3.13) r_{i-1} есть последнее уже сгенерированное псевдослучайное число, r_i — псевдослучайное число, которое будет вычислено, a, m, n — константы, а r_0 — начальное число в последовательности, называемое *начальным значением*. Начальное значение должно быть задано пользователем и определяет все числа в последовательности. Так как $0 \leqslant r_i \leqslant 2^n-1$, то, чтобы получить число y_i между 0 и 1, мы должны разделить r_i на 2^n.

Если последовательности чисел, используемые в стохастических моделирующих программах для целей, перечисленных выше, не обладают достаточной степенью *независимости* и *равномерности*, результаты моделирования могут привести к большим ошибкам. К используемым последовательностям можно (и нужно) применять различные статистические тесты независимости и равномерности (см. Кнут (1969)).

Второй шаг процедуры требует решения уравнения $y^*=F_X(x^*)$. Если функция $F_X(x)$ задана в *эмпирической форме* как таблица пар $(x, F_X(x))$, значения x^* могут быть получены интерполяцией. Однако $F_X(x)$ можно задать и в *аналитической форме*, которая получается при сглаживании измеренного распределения или при введении исследователями упрощающих предположений. Если аналитическая форма существует также для обратной функции $x=F_X^{-1}(y)$, отображение псевдослучайных чисел на горизонтальную ось можно осуществить аналитически, без интерполяции. Например, если X распределено экспоненциально, мы имеем

$$y=F_X(x)=1-e^{-x/\overline{X}} \tag{3.14}$$

и

$$x=F_X^{-1}(y)=-\overline{X}\ln(1-y). \tag{3.15}$$

Таким образом, для любого заданного y^* уравнение (3.15) дает соответствующее значение x^*.

Аналогичный подход можно использовать для получения псевдослучайных чисел, подчиненных заданному дискретному распределению. Пусть X — случайная переменная, принимающая только два значения: x_1 с вероятностью p_1 и x_2 с вероятностью $p_2=1-p_1$. Заменим шаг (b) приведенной выше процедуры следующим правилом:

$$\begin{aligned} x^*&=x_1, \text{ если } 0 \leqslant y^* < p_1, \\ x^*&=x_2, \text{ если } p_1 \leqslant y^* < 1. \end{aligned} \tag{3.16}$$

Распределение для случайной величины X^*, определенной соотношениями (3.16), получается из соотношений

$$\begin{aligned} \text{Prob}\,(X^*=x_1)&=\text{Prob}\,(0 \leqslant y^* < p_1)=p_1, \\ \text{Prob}\,(X^*=x_2)&=\text{Prob}\,(p_1 \leqslant y^* < 1)=1-p_1=p_2. \end{aligned} \tag{3.17}$$

Таким образом, X^* имеет то же самое распределение, что и X. Эту процедуру можно использовать, например, для определения следующего состояния, когда рассматриваемая в этом примере система покидает S_1 — единственное состояние, из которого можно попасть в несколько следующих состояний. Интервал (0,1) делится на

подынтервалы, длины которых равны вероятностям $p_{бр}$, $p_{дс}$ и $p_{оп}$. Генерируется псевдослучайное число y^*, и следующее состояние выбирается в зависимости от того, какой подынтервал содержит y^*:

$$\begin{aligned} &\text{если } 0 \leqslant y^* < p_{бр}, \text{ то } S_1 \longrightarrow S_2; \\ &\text{если } p_{бр} \leqslant y^* < (p_{бр}+p_{дс}), \text{ то } S_1 \longrightarrow S_3; \\ &\text{если } (p_{бр}+p_{дс}) \leqslant y^* < 1, \text{ то } S_1 \longrightarrow S_0. \end{aligned} \tag{3.18}$$

Таким образом, переходы из состояния S_1 в состояния S_2, S_3 и S_0 будут происходить с вероятностями $p_{бр}$, $p_{дс}$ и $p_{оп}=1-p_{бр}-p_{дс}$ соответственно.

В более общем случае этот же подход можно применить к *стохастическим моделям с дискретными состояниями*. Стохастическая модель с дискретными состояниями

Таблица 3.4

Полумарковская программная модель для примера 3.3

(a) *Матрица вероятностей переходов*

	В состояние S_0	S_1	S_2	S_3
Из состояния S_0	0	1	0	0
S_1	$p_{оп}=1-p_{бр}-p_{дс}$	0	$p_{бр}$	$p_{дс}$
S_2	0	1	0	0
S_3	0	1	0	0

(b) *Распределения времен пребывания в состояниях*

Состояние	Время пребывания (случайная переменная)
S_0	$t_{своб}$
S_1	$t_{цп}$
S_2	$t_{бр}$
S_3	$t_{дс}$

(c) *Начальное состояние*

S_3

(также называемая *цепью*) есть стохастический процесс с конечным или счетным множеством состояний. В нашем примере [см. рис. 3.3(b)], мы моделировали программу процессом, который в каждый данный момент времени находится в одном из четырех состояний. Если мы зададим распределения времен пребывания в этих состояниях — вероятности того, что в момент перехода данное состояние заменится на каждое из остальных состояний, — и начальное состояние, мы полностью зададим стохастическую модель этой программы (см. табл. 3.4). Как будет объяснено в разд. 4.3.1, это *полумарковская модель с непрерывным временем*.

Процедура, посредством которой может быть реализована модель в табл. 3.4, должна убедить читателя в ее всеобщей применимости. Прогон начинается с приведения модели в начальное состояние, например S_0. Затем по распределению $t_{свободен}$ определяется продолжительность пребывания в состоянии S_0. Когда этот интервал истекает, должен произойти переход. Строка матрицы вероятностей переходов в табл. 3.4(a), соответствующая текущему состоянию, представляет собой дискретное распределение для выбора следующего состояния. Так как в этом случае первая строка в столбце S_1 содержит 1, а во всех остальных — нули, должен произойти переход в S_1.

Когда интервал центрального процессора истечет, мы с помощью строки S_1 снова выберем следующее состояние из трех возможных кандидатов и т. д. ■

Моделирующая программа может быть сконструирована таким образом, чтобы при минимальных локальных модификациях, она могла работать с описанием рабочей нагрузки в любой из введенных выше трех форм или в любой их комбинации. Разумеется, любые изменения в описании рабочей нагрузки, вообще говоря, требуют более глубоких и глобальных изменений общей структуры моделирующей программы, а также отдельных ее частей. Очень гибкая организация, по-видимому, минимизирующая эти изменения, приведена в разд. 8.4. В данном же разделе мы коснемся только представления модели рабочей нагрузки, которое можно обсуждать отдельно от моделей систем. Выбор модели рабочей нагрузки, который тесно связан со структурой моделирующей программы и степенью детальности, обсуждается в гл. 5.

Пример 3.4. Модель рабочей нагрузки для моделирующей программы, описанной в примере 3.2, должна содержать всю информацию, необходимую для многочисленных решений, которые должна принять моделирующая программа, чтобы воспроизвести прохождение каждого задания по карте событий рис. 3.8. Пусть запросы ресурсов для заданий вычисляются, как показано в табл. 3.5. Тогда моделирующей программе должны быть заданы (или она сама должна определить) значения величин, перечисленных в табл. 3.6.

Таблица 3.5

Формулы, используемые моделирующей программой из примера 3.2 для вычисления запроса ресурсов

$t_{уч} = c/v_{уч}$	$t_{цп} = t_{цп}$
$t_{дс} = \overline{t}_{ддс} + t_{сдс}\, r_{дс}$	$t_{уп} = l/v_{уп}$
$t_{бр} = \overline{t}_{дбр} + t_{сбр}\, r_{бр}$	$m = m$

Символ	Единица измерения	Определение
c	карт	Число карт во входном потоке
l	строк	Число строк для вывода на печать
m	слов	Запрошенный объем памяти
$r_{бр}$	слов	Запрошенное количество слов в записи на барабане
$r_{дс}$	слов	Запрошенный размер записи на диске в словах
$\overline{t}_{дбр}$	мс	Среднее время доступа к барабану
$\overline{t}_{ддс}$	мс	Среднее время доступа к диску
$t_{цп}$	мс	Запрошенное непрерываемое время ЦП
$t_{уч}$	мин	Запрошенное время устройства чтения перфокарт
$t_{бр}$	мс	Запрошенное время барабана
$t_{дс}$	мс	Запрошенное время диска
$t_{уп}$	мин	Запрошенное время устройства печати
$t_{сбр}$	мс/слово	Время обмена с барабаном
$t_{сдс}$	мс/слово	Время обмена с диском
$v_{уч}$	карт/мин	Скорость чтения карт устройством чтения перфокарт
$v_{уп}$	строк/мин	Скорость вывода на печать

Таблица 3.6

Информация о рабочей нагрузке, необходимая для моделирующей программы из примера 3.2

	Символ	Определение
Для каждого задания	$t_{пост}$	Время поступления задания
	c	Число карт во входном потоке
	l	Число строк, выводимых на печать
	n	Число шагов задания
Для каждого шага задания	m	Запрошенный объем памяти
	$n_{цп}$	Число непрерываемых интервалов времени ЦП
	$n_{бр}$	Число запросов барабана
	$n_{дс}$	Число запросов диска
Для каждого запроса ресурса	$t_{цп}$	Запрошенное непрерываемое время ЦП
	B	Функция-индикатор, определяющая, будет ли запрошен барабан или диск в конце каждого интервала ЦП
	$r_{бр}$	Число слов при обмене с барабаном
	$r_{дс}$	Число слов при обмене с диском

Заметим, что в эту таблицу входит время поступления задания $t_{пост}$. Аналогичны времена приема данных с клавиатуры, появляющиеся в карте событий на рис. 3.8, предполагаются параметрами скорее системы, чем рабочей нагрузки. Это предположение соответствует нашему обсуждению границ вычислительной системы в разд. 1.2.

Мы предполагаем, что моделирующая программа оценит по параметрам задания c и l, сколько раз каждое задание должно воспользоваться соответственно входным и выходным буферами. Числа $n_{цп}$, $n_{бр}$ и $n_{дс}$ не являются независимыми, так как они должны удовлетворять ограничению $n_{бр} + n_{дс} = n_{цп} + 1$. Из-за различных стратегий распределения ресурсов могут потребоваться дополнительные параметры рабочей нагрузки, например приоритеты заданий, время выполнения задания по оценке пользователем и предельный размер памяти.

Таким образом, моделирующей программе должны быть заданы значения величин, перечисленных в табл. 3.6, или данные, из которых можно вычислить эти значения. Так и окажется впоследствии, когда понадобится каждый элемент информации. Возможным способом выполнения этой функции в моделирующей программе было бы наличие всех данных, характеризующих задание, уже считанными или сгенерированными к моменту поступления задания. Эти данные должны затем запоминаться в соответствующих ячейках базы данных моделирующей программы, где подпрограммы моделирующей программы могут отыскать их при необходимости, и выбрасываться только тогда, когда задание покидает систему. Заметим, что такая организация, которая подразумевалась еще при определении задач программ событий ПОСТУПЛЕНИЕ (J) и УХОД (J) в примере 3.2, позволяет одной и той же моделирующей программе работать под управлением различных представлений рабочей нагрузки. Разумеется, это только одна из нескольких возможных организаций. При другом устройстве описания шага задания могли бы генерироваться в момент инициирования шага задания.

Читатель, конечно, понимает, что любое из представлений рабочей нагрузки, описанных в начале этого раздела, может обеспечить моделирующую программу необходимой информацией.

Представление событий в форме трассировочной записи содержит значения параметров, перечисленных в табл. 3.6, кроме, возможно, n, $n_{цп}$, $n_{бр}$, $n_{дс}$ и B, которые можно вывести из самой трассировочной записи и из порядка запросов. Этот порядок

позволяет моделирующей программе определить хронологическую последовательность, в которой различные ресурсы запрашиваются каждым заданием. Обычно трассировочная запись для системы состоит из ряда записей отдельных событий, относящихся к различным заданиям, и содержит смесь сведений о рабочей нагрузке и о системе. Такая запись должна быть обработана предварительно, чтобы ее можно было использовать в качестве входной информации для моделирующей программы, управляемой трассировочной записью. Кроме того, степень детальности, с которой трассируется поведение системы, должна быть достаточна для удовлетворения информационных потребностей моделирующей программы.

Представление рабочей нагрузки в программной форме обычно состоит из ограниченного числа программ, которые должны создавать для модели системы нагрузку, подобную реальной рабочей нагрузке реальной системы. Таким образом, это представление по своей природе предполагает использование моделей рабочей нагрузки, концептуально аналогичных тем, которые используются в управляемых измерительных экспериментах. Эти модели, на которых основываются выполняемые искусственные рабочие нагрузки, описаны в разд. 5.3.4 и 5.3.5. Программы из модели рабочей нагрузки могут быть использованы программой ПОСТУПЛЕНИЕ для генерации описания задания в табл. 3.6.

Если моделирующая программа относится к типу управляемых распределением, существует несколько возможностей, характеризуемых разными степенями детальности. Например, каждое задание могло бы иметь собственный набор распределений для параметров шага задания и запросов, тогда как параметры задания $t_{\text{пост}}$, c, l и n были бы распределены идентично. Простейшее и наиболее грубое решение состоит в назначении моделирующей программе глобальных распределений следующих параметров: t_{int} (интервал между прибытиями последовательных заданий), c, l, n, m, $n_{\text{цп}}$, $t_{\text{цп}}$, $r_{\text{бр}}$ и $r_{\text{дс}}$. Значение $b_{\text{бр}}$ вероятности запроса на барабан при окончании интервала времени центрального процессора должно быть также назначено. Когда оканчивается $n_{\text{цп}}$-й интервал использования центрального процессора данного шага задания, выдается запрос на диск, а после удовлетворения этого запроса пространство в памяти, назначенное этому шагу задания, освобождается. Эти распределения будут использованы моделирующей программой так, как это описано в примере 3.3. Псевдослучайные числа, соответствующие данным распределениям, могут быть вычислены программой ПОСТУПЛЕНИЕ в момент поступления задания, так чтобы генерировать полное описание задания. Иначе они могли бы быть найдены любой программой обработки события, которая нуждается в некоторых недоступных в данный момент значениях параметров задания. ■

Какой тип представления следует принять для модели рабочей нагрузки в работе по имитационному моделированию вычислительной системы? Решение с помощью модели, управляемой распределением, вообще говоря, является единственным, требующим наиболее компактного описания рабочей нагрузки. Однако компактные описания часто не очень точны. Например, принимая это решение, обычно пренебрегают действительно существующими корреляциями между параметрами задания, так как распределения считаются независимыми. Корреляции можно было бы учитывать, но это заметно увеличило бы количество необходимой информации.

Проблема, которой никогда не следует пренебрегать, это возможность получения данных. Не только корреляции относятся к обычно недоступным и трудно измеримым параметрам, но даже формы и параметры распределений для таких важных величин, как t_{int}, m, $t_{\text{цп}}$, трудно отыскать в литературе. Даже если это можно сделать, эти распределения очень трудно применить к установкам, отличающимся от тех, на которых они были измерены. Таким образом, очень часто наи-

более реальным решением оказывается измерение необходимых распределений на реальной установке, если она существует.

Некоторые из упомянутых выше проблем просто не возникают для моделирующей программы, управляемой трассировочной записью. Например, корреляции неявно содержатся в трассировочной записи. Поэтому их не надо измерять или моделировать явно. Два недостатка трассировочных записей, которые в ряде случаев могут оказаться довольно серьезными, это их большой объем и невозможность модификации. Представление рабочей нагрузки в виде трассировочной записи создает неудобства при моделировании изменений рабочей нагрузки, не встречавшихся в реальных трассировочных записях, тогда как изменение распределений в программе, управляемой распределением, сделать просто.

Решение с помощью программно-управляемой модели выдвигает проблемы, с которыми обычно сталкиваются при конструировании искусственных рабочих нагрузок. Оно обычно более компактно и более гибко, чем управляемое трассировочной записью, но существенно менее точно. В частности, разнообразие последовательностей продолжительностей пребывания в различных состояниях, в программно-управляемой модели, обычно весьма ограничено, если только не используются линейные программы. Однако в этом случае, они весьма напоминают трассировочные записи и утрачивают всю свою компактность. По отношению к решению, управляемому распределениями, представление в форме программы, по-видимому, менее гибко и менее компактно. Его точность может оказаться выше, но только если программа сконструирована весьма тщательно.

Примеры программ моделирования вычислительных систем, управляемых трассировочной записью, описаны в работах Ченга (1969), Бейлнера и Вальдбаума (1972), Ноу и Натта (1972), Шермана и др. (1972). Решения, с помощью программно-управляемых моделей, использовались в работах Нильсена (1967) и Симена и Соуси (1969). Работа Мак-Дугалла (1970) описывает один из многих примеров моделирующей программы, управляемой распределением.

3.2.5. Реализация моделирующей программы

Выбор языка программирования для написания моделирующей программы является, как было сказано в разд. 3.2.1, одним из основных решений, принимаемых при планировании имитационного исследования. Приемлемые языки относятся к одной из трех категорий:

a) языки моделирования;

b) общие языки, не приспособленные специально для моделирования;

c) расширения общих языков, ориентированные на моделирование.

Категория «а» включает такие языки, как GPSS [см. Гордон (1975)], SIMSCRIPT [см. Кивиат и др. (1969)] и СИМУЛА [см. Дал и Нигард

(1966)]. Категория «b» включает язык ассемблера, ФОРТРАН, АЛГОЛ и ПЛ/I. Категория «c» включает язык GASP [см. Притскер (1974)], являющийся расширением ФОРТРАНа. Часто, как и в случае GASP, расширение общего языка, переводящее его в категорию «c», заключается в добавлении пакета подпрограмм к библиотеке существующего компилятора.

Языки моделирования ориентированы не только на помощь программистам в кодировании моделирующей программы, но также и на обеспечение их структурой, в которой они могут гораздо легче формулировать свои модели. Это так называемый *универсальный подход* моделирования, который приходится разделять их пользователям, если они намерены пользоваться этими языками. Этот универсальный подход оказывает глубокое влияние на организацию моделирующих программ, написанных на этом языке. SIMSCRIPT основан на подходе с планированием событий (см. пример 2.2), а GPSS и СИМУЛА — на подходе с взаимодействием процессов. Если быть более точным, GPSS структурирует моделирующую программу в соответствии с тем, что было названо в разд. 3.2.2 версией 1 этого подхода, а СИМУЛА — в соответствии с версией 2.

В общем случае язык моделирования обладает особенностями, облегчающими

a) статическое описание системы, которую предстоит моделировать (операторы создания и уничтожения объектов и спецификации их атрибутов и взаимосвязей между ними);

b) динамическое описание системы (например, автоматическая установка часов времени модели, определение и планирование событий);

c) представление стохастических явлений (генерация псевдослучайных чисел, генерация выборок из распределений, заданных в аналитической или эмпирической формах);

d) сбор, сжатие и представление данных (т. е. все задачи, относящиеся к средствам моделирующей программы).

Другие полезные особенности, которые можно найти в некоторых языках, включают средства контроля модели, отладки моделирующей программы (особенно трудная задача, если модель стохастическая) и приведения ее в желаемое начальное состояние (см. разд. 3.4.2). Конечно, наличие в языке моделирования понятий и конструкций, представляющих центральные концепции моделирования, является весьма полезным, особенно для начинающего. Однако некоторые исследователи могут воспринимать эти понятия и конструкции как ограничения. В некоторых языках, например, может вызвать затруднение написание моделирующей программы, управляемой не распределением, или какой-либо необычной схемы управления очередью. Вообще говоря, есть все основания ожидать, что использование языка моделирования уменьшит затраты времени на кодирование и отладку моделирующей программы. Кроме того, языки моделирования представляют

собой гораздо лучшее средство описания системы, чем другие языки. Это делает их более мощным средством коммуникации, а моделирующие программы, написанные на них, более легкими в отладке. Однако эти моделирующие программы часто менее эффективны при выполнении, поскольку эффективность объектного кода не относится обычно к основным целям при разработке языка моделирования. Другие частые жалобы на большинство существующих языков моделирования относятся к их средствам отладки, документации и транспортабельности. В большинстве случаев эти недостатки, по-видимому, возникли исторически и не являются непреодолимыми, так что их не следует считать дискредитирующими языки моделирования вообще. Однако этим можно объяснить, почему пока эти языки не настолько широко распространены, насколько это могло бы быть.

Расширения общих языков представляют промежуточное решение. Они могут быть легко освоены программистами, знакомыми с общим языком, в который вложены эти расширения. Они обеспечивают универсальный подход и средства программирования (структуры данных, механизмы управления моделированием, стохастические программы, средства обслуживания), аналогичные перечисленным выше для языков моделирования. Кроме того, они, вообще говоря, создают более эффективную кодировку программы, чем языки моделирования и создают более транспортабельные моделирующие программы. Однако синтаксические ограничения и особенности структуры общего языка, на котором они основаны, могут сделать модель менее удобочитаемой и менее понятной, чем при использовании языка моделирования.

Как выбрать язык для написания программы, моделирующей вычислительную систему? Во многих практических ситуациях альтернативы не существует. Доступность языка является на практике наиболее важным критерием, а число установок, располагающих процессором языка моделирования (или расширением общего компилятора, ориентированным на моделирование), хотя и увеличивается, но все еще относительно мало. Это определенно одна из главных причин, почему в большей части существующих вычислительных систем используются общие, а не специализированные языки.

Другой важный довод состоит в более высокой эффективности (по времени выполнения), обычно обеспечиваемой общими языками, что особенно важно, если ожидается, что предпринимаемое исследование может потребовать большого количества прогонов модели.

Третий довод состоит в нежелании некоторых программистов изучать новый язык, если они чувствуют, что знакомый им язык может удовлетворительно использоваться для решения данной задачи. Однако упрощение формулировки модели, легкость написания, отладки и сопровождения, оказывающие существенное влияние на стоимость и эффективность моделирующей программы, являются аргументами в пользу выбора языка моделирования или по крайней мере соответствующего расширения общего языка. Действительно, известны случаи, когда моделирующая программа была написана, отлажена и калибро-

вана на языке моделирования, а затем ее производственная версия была переписана на общем языке (ассемблера или высокого уровня). Расширения общих языков иногда являются, как можно было заметить из изложенного выше, хорошим компромиссом между преимуществами и недостатками остальных двух типов языков программирования для написания моделирующих программ.

Обсуждение принципов, специфических особенностей и относительных достоинств существующих языков моделирования выходит за рамки этой книги. Мы рекомендуем заинтересованному читателю обратиться к уже опубликованным обзорам и сравнительным исследованиям [см. например, Тейчроев и Любин (1966), раздел IX. 3.1 книги Саммет (1969), Кивиат (1971) и гл. 4 и 5 книги Фишмана (1973)]. Использование языка ФОРТРАН для написания модели вычислительной системы иллюстрируется, например, Мак-Дугаллом (1970).

3.3. КАЛИБРОВКА И ПРОВЕРКА ПРИГОДНОСТИ МОДЕЛИ

3.3.1. Постановка задачи

Основной вопрос, возникающий при использовании метода моделирования, состоит в том, насколько хорошо модель представляет моделируемую систему. Только удовлетворительный ответ на этот вопрос может придать результатам исследования, основанного на методе моделирования, необходимую достоверность. Модель, поведение которой слишком отличается от поведения моделируемой системы, практически бесполезна. Но каково допустимое различие?

Как было сказано в начале разд. 3.1, можно определить *точность* модели в терминах индексов производительности, выбранных для оценки системы. Модель достаточно точна, если значения полученных на ней индексов отличаются от индексов, полученных на моделируемой системе, менее чем на заданные величины максимальных ошибок. Таким образом, точность модели измеряется разностями между производительностями системы и ее модели. Чтобы упростить наши рассмотрения, предположим, что измерения индексов производительности существующей системы, производимые для сравнения с индексами модели, не подвержены ошибкам. Однако из гл. 2 мы знаем, что это, вообще говоря, неверно.

Это определение точности схематически иллюстрируется рис. 3.16. Система $\mathcal{S}$ при рабочей нагрузке W имеет производительность $\mathcal{P}$. Индексы производительности, значения которых в совокупности обозначены $\mathcal{P}$, могут быть скалярами, скалярными функциями параметров установки, распределениями или последовательностями значений величин, рассматриваемых как случайные переменные, а также описателями распределений, такими, как средние и дисперсии.

$\mathcal{S}'$ — это модель $\mathcal{S}$. Управляемая нагрузкой W', которая моделирует W, $\mathcal{S}'$ показывает производительность $\mathcal{P}'$. Сравнение значений одноименных индексов в $\mathcal{P}$ и $\mathcal{P}'$ дает меру точности $\mathcal{S}'$ и W'. Рассмотрим в качестве примера модель диалоговой APL-системы, имеющую вид уравнения регрессии (2.52). Мерой ее точности будет сумма квадратов ошибок, обозначенная через SSE в табл. 2.22. Чем меньше отношение SSE к общей сумме квадратов SST (или, что то же самое, чем больше коэффициент множественной корреляции R^2), тем выше точность модели.

Трудности возникают тогда, когда моделируемая система не существует физически или недоступна для эксперимента. Это как раз тот

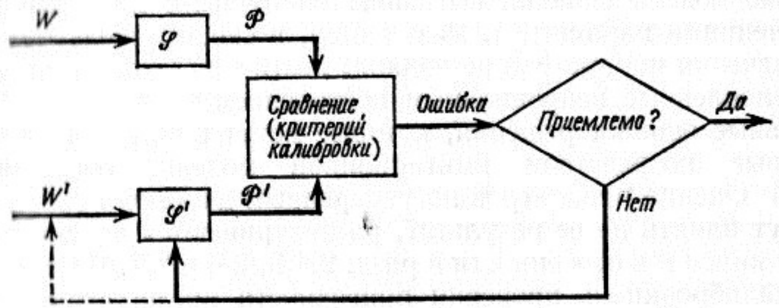


Рис. 3.16. Блок-схема итеративной процедуры калибровки.

случай, когда методы моделирования применяются к проблемам конструирования. Это препятствие можно преодолеть, признав, что моделируемая система является в действительности концептуальной моделью конструируемой системы. Таким образом, модель используется в данном исследовании для того, чтобы представлять эту концептуальную модель, существующую пока в умах конструкторов и в их письменных спецификациях. С практической точки зрения, убедиться, что концептуальная модель корректно оттранслирована в моделирующую программу или в аналитическую модель, может оказаться не легкой задачей. Эта проблема в принципе похожа на проверку корректности программы, и к ней можно подойти с теми же методами. В частности, мы можем выбрать ряд тестовых случаев, использовать модель для получения значений индексов производительности и проверить разумность результатов моделирования и их соответствие ручным просчетам.

Если точность модели неудовлетворительна, то модель должна быть изменена, а процесс проверки повторен. Эта операция называется *калибровкой*. Итеративная процедура калибровки схематически показана на рис. 3.16. Процедура оканчивается, как только исследователи найдут точность модели удовлетворительной.

Предполагая, что значение $\mathcal{P}$ на рис. 3.16 не подвержено влиянию ошибок, мы должны отнести все неточности, обнаруженные при сравнении $\mathcal{P}$ и $\mathcal{P}'$, на счет модели. Существуют три основных класса ошибок, воздействующих на $\mathcal{P}'$:

a) *ошибки формулировки*, обусловленные ошибочной или неполной, или недостаточно подробной моделью;

b) *ошибки решения*, обусловленные некорректным или чрезмерно упрощенным методом решения модели;

с) *ошибки задания параметров*, обусловленные использованием неверных значений параметров.

Цель калибровки состоит в уничтожении или уменьшении неточностей формулировки. Однако точная модель может давать неверные результаты даже в процессе калибровки, если метод решения или значения ее параметров или и то и другое неверны. Таким образом, конструктор или пользователь модели должен избегать приписывания поведению модели ошибок, вызванных неточными методами решения или значениями параметров. Разумеется, в свою очередь метод решения и значения параметров не должны нести ответственности за ошибки, обусловленные неправильным построением модели.

Основные ошибки решения, которые могут влиять на результаты, полученные посредством имитационной модели, обсуждаются в разд. 3.4. Оценка параметров по измеренным данным и ошибки, которые могут влиять на ее результат, рассматриваются во многих разделах этой книги и в особенности в разд. 2.4.3, 2.6 и 2.7. В нашем обсуждении калибровки и проверки пригодности предположим для простоты, что единственными ошибками, влиянием которых на значения $\mathcal{P}'$ нельзя пренебречь, являются ошибки формулировки модели.

Реализация простой схемы, изображенной на рис. 3.16, выдвигает ряд важных проблем. Одна из них — как проверить, что модель рабочей нагрузки правильно представляет действительную рабочую нагрузку. Этот существенный и трудный вопрос обсуждается в гл. 5. Другая проблема, рассматриваемая в разд. 3.3.3, относится к критериям сравнения производительностей модели и системы. Это может оказаться затруднительным, если индексы производительности представлены не несколькими скалярными величинами, а распределениями случайных величин. Статистические методы сравнения обсуждаются в разд. 6.3.1.

Выбор индексов и значений максимальных допустимых ошибок зависит от типа выполняемого исследования. В работах, включающих сравнение различных систем или различных версий одной системы, на самом деле существенны не точные значения индексов производительности, а их чувствительность к рассматриваемым изменениям. Пусть $\mathcal{S}^*$ — система, сравниваемая с $\mathcal{S}$, и пусть $\mathcal{P}^*$ — производительность, показанная системой $\mathcal{S}^*$ при рабочей нагрузке W. Предположим для простоты, что $\mathcal{S}^*$ есть измененная версия системы $\mathcal{S}$ и что мы интересуемся единственным скалярным индексом производительности. Изменив модель $\mathcal{S}'$ таким образом, чтобы отразить различие между $\mathcal{S}$ и $\mathcal{S}^*$, мы получим модель $\mathcal{S}^{*\prime}$, показывающую производительность $\mathcal{P}^{*\prime}$. Ясно, что в этом случае важнее всего сравнивать $\mathcal{P}^*-\mathcal{P}$ и $\mathcal{P}^{*\prime}-\mathcal{P}'$. Иметь модели, которые, кроме того, точно воспроизводят значения данного индекса, разумеется, достаточно, но никоим образом не необходимо в исследовании данного типа.

Еще одна проблема возникает из того очевидного факта, что сравнения не могут быть исчерпывающими. Модель нельзя откалибровать

для всех экспериментальных условий, в которых она должна использоваться. Это практически невозможно из-за, вообще говоря, большого количества таких условий и их непредсказуемости. Более того, если бы была проделана исчерпывающая калибровка с данными измерений, модель оказалась бы бесполезной, поскольку больше бы не потребовалось предсказывать производительность системы. Модель обычно используется в условиях, в которых ее точность не была, а часто и не могла быть проверена. Таким образом, калибровка обычно ограничена несколькими комбинациями параметров рабочей нагрузки и системы. Если система существует и ее можно измерить, эти комбинации следует выбрать так, чтобы эти эксперименты были выполнимы и по возможности просты. Модель во время калибровки должна управляться входными переменными, представляющими выбранные комбинации системных параметров и параметров рабочей нагрузки.

При заданных определенных требованиях к точности в терминах максимальных допустимых ошибок, мы говорим, что модель *пригодна* для некоторого входного условия, если она удовлетворяет этим требованиям, когда ее входные переменные соответствуют этому условию. *Область пригодности модели* есть множество входных условий, для которых пригодна модель. Разумеется, размер этой области зависит от того, насколько строги требования точности. При калибровке модели мы добиваемся ее пригодности для входных условий, выбранных для калибровки. Как можно определить, пригодна ли модель также и к тем условиям, которые встретятся ей при ее практическом использовании? Существует ли метод проверки *устойчивости* модели, определяемой как степень нечувствительности ее точности к изменению входных условий?

Ответить на все эти вопросы было бы можно, если бы была известна пригодная всегда *процедура проверки*. К сожалению, это не так. Поэтому создатели моделей вынуждены прибегать к методам «для данного случая», частичным тестам, а кроме всего, к здравому смыслу. Некоторое ощущение устойчивости модели можно получить, выполняя после ее калибровки немного дополнительных, тщательно отобранных сравнений типа показанных на рис. 3.16 с использованием новых экспериментальных данных. Если доступные данные нельзя дополнить новыми, оценщики могут попытаться увеличить надежность модели, используя часть данных для ее калибровки, а остаток — для частичной проверки. Часто бывает также полезна апостериорная проверка. Она состоит в сравнении предсказаний модели и результатов измерений после того, как будет произведено какое-либо изменение в установке, предусмотренное и в исследовании. Если модель окажется приемлемо точной, уверенность пользователей в ее устойчивости оправданно возрастет.

Другая, даже более важная проблема состоит том, как конструировать достаточно устойчивые модели. В общем случае можно следовать принципу: чем ближе структура модели к структуре системы и чем выше ее степень детальности, тем обширнее область пригодности модели. К сожалению, очень подробная модель весьма дорога.

Степень устойчивости модели можно повысить тщательной калибровкой. В частности, мы можем выбрать более сильные критерии калибровки, чем строго необходимый. Предположим, например, что мы выбрали в качестве индекса производительности время прохождения задания. Если в качестве *критерия калибровки* мы выберем некоторую меру различия между распределением времен прохождения в модели и в системе (вместо разности средних значений), модель, которую мы получим, вероятно, будет иметь более широкую область пригодности. Можно ожидать, что устойчивость модели улучшится, если критерий калибровки будет основан на времени прохождения каждого задания

Таблица 3.7

Критерии калибровки модели (индекс производительности: время прохождения задания)

Основание критерия	Ошибка E
Среднее время прохождения задания	$\frac{1}{N}\sum_{j=1}^{N}(t_j-t'_j)$
Время прохождения отдельного задания	$\sum_{j=1}^{N}(t_j-t'_j)^2$
Продолжительность отдельного шага задания	$\sum_{j=1}^{N}\sum_{i=1}^{n_j}(t_{ji}-t'_{ji})^2$

Символ	Определение
t_j	Время прохождения задания j в системе
t'_j	Время прохождения задания j в модели
t_{ji}	Время выполнения шага i задания j в системе
t'_{ji}	Время выполнения шага i задания j в модели
N	Общее число заданий
n_j	Число шагов в задании j

или, быть может, на продолжительностях шагов задания вместо распределения этого времени по всем заданиям. Определения ошибок, соответствующих этим трем критериям, можно выразить так, как это сделано в табл. 3.7. Эти критерии вновь будут обсуждаться в разд. 3.3.3.

В общем случае нам следует ожидать, что увеличение степени детальности не ухудшит устойчивость моделирующей программы. Иногда, однако, в процессе калибровки существует точка, за которой устойчивость может ухудшиться из-за чрезмерной *подстройки* модели к частным входным условиям, использованным для ее калибровки. К сожалению крайне трудно определить момент, когда эта точка достигнута.

Калибровку можно рассматривать как итеративный процесс оптимизации, в котором показатель качества (точность модели) должен максимизироваться. Итеративная калибровка экономически оправдана, если цена использования модели не слишком высока. Например, в случае имитационной модели время прогона модели не должно быть слишком велико. Иными словами, моделирующая программа не должна быть слишком детальной. Заметим, что этот тип моделирующей программы обычно оказывается как раз тем, который требует наиболее тщательной калибровки. Скорость сходимости к калиброванной модели, другой важный аспект итеративной процедуры, зависит от «расстояния» начальной модели от калиброванной и от метода, используемого для определения очередного изменения, которое следует внести в ходе процедуры. Моделирующая программа должна быть достаточно гибкой, чтобы изменения, предусмотренные процедурой калибровки, можно было вносить относительно легко. Эти изменения обсуждаются в следующем разделе. Дальнейшие соображения о критериях калибровки и о выборе изменений на каждой итерации можно найти в разд. 3.3.3.

3.3.2. Калибровка моделирующей программы

Для калибровки моделирующей программы можно производить изменения трех типов:

a) глобальные изменения (например, добавление программ обработки событий, изменение событий, планируемых определенными программами);

b) локальные изменения (например, замена моделей компонентов внешне эквивалентными, но более точными моделями);

c) изменения некоторых специальных параметров, называемых *калибровочными*.

Эти типы изменений иллюстрируются следующим примером.

Пример 3.5. Предположим, что модель ПРОП-системы пакетной обработки, которую мы сконструировали в примерах 3.2 и 3.4, уже написана и отлажена. Прежде чем использовать модель, мы должны удостовериться в ее точности и, если необходимо, улучшить ее. При формулировке модели мы сделали ряд упрощающих предположений. Мы решили, например, игнорировать конфликты доступа к памяти и накладные расходы операционной системы, а также представили диск и барабан весьма грубыми аналитическими моделями (см. табл. 3.5). Разумны ли эти предположения в контексте нашего исследования? Является ли их воздействие на точность результатов допустимым? Каковы последствия пренебрежения рядом деталей системы и рабочей нагрузки, кроме уже упомянутых выше?

Если при первых калибровочных прогонах обнаруживаются заметные различия между результатами моделирования и имеющимися результатами измерений, мы, естественно, приходим к предположению, что эти различия обусловлены неточностями нашей аппроксимации. Мы можем, например, сформулировать гипотезу: недостаток точности модели обусловлен в значительной мере тем, что не были приняты во внимание накладные расходы. Экспериментальные данные о величине накладных расходов, если они доступны, могут либо поддержать эту гипотезу, либо опровергнуть ее. Модификация модели для учета накладных расходов требует ряда глобальных и локальных изменений. Таким образом, было бы полезно получить некоторую дополнительную информацию о вероятности того, что эта догадка правильна, прежде чем предпри-

нимать громоздкую модификацию. Приближения, кажущиеся нам весьма грубыми, могут на самом деле не оказывать существенного влияния на точность модели.

В нашем случае мы могли бы модифицировать описание рабочей нагрузки различными способами для грубого определения порядка величины влияния накладных расходов на среднюю пропускную способность системы. Например, предположив, что модель управляется распределением, мы можем заметно увеличить среднее распределения времени ЦП и посмотреть, какое воздействие окажет это увеличение на выходные данные модели. При выборе этого увеличения мы можем ограничиться тем, чтобы оно по возможности превышало действительный прирост работы ЦП, обусловленный накладными расходами. Если соответствующее изменение средней пропускной способности не значимо при разумном доверительном уровне, мы решим, что наше предположение неверно, а накладные расходы можно на самом деле игнорировать без ущерба для точности. Это заключение, однако, основывается на результатах эксперимента «проверка факторов по одному». Таким образом, оно могло бы оказаться неправильным при других входных условиях.

Даже взаимодействие приближений могло бы сделать наши выводы неверными. В принципе возможно, что два упрощения порознь не понижают заметно точность модели, а вместе понижают. Проверка наших гипотез должна тщательно планироваться так, чтобы минимизировать вероятность того, что их результаты приведут нас к неправильным выводам. В частности, за экспериментами «проверка факторов по одному», обычно дающими полезную, легко интерпретируемую информацию, должны следовать более сложные эксперименты, в которых можно определить эффекты комбинаций приближений.

Подходы, которые можно использовать для решения, следует ли увеличивать степень детальности модели и как это сделать, могут также применяться к проблеме упрощения структуры калиброванной модели с целью уменьшения времени выполнения. Эксперименты, о которых говорилось выше, можно в действительности провести для определения эффектов предполагаемого упрощения. Однако часто советуют оставлять модель несколько более подробной, чем требуется для работы, чтобы не ограничивать слишком ее область пригодности.

Большая часть сделанных выше замечаний применима также к изменениям локальной степени детальности модели. Типичная модификация этого типа — замена, скажем, дисковой программы на другую, с иной степенью детальности. Например, программа, вычисляющая $t_{дс}$ согласно табл. 3.5, может быть заменена на программу, основанную на модели (3.5). Как можно определить, какие подпрограммы следует изменить для улучшения точности модели? Метод, описанный выше для оценки эффектов глобальных изменений, можно использовать также для локальных модификаций. В частности, можно использовать эксперимент типа «проверка факторов по одному» с учетом обсужденных выше предостережений и ограничений.

Обычно количество деталей, которые нужно добавлять, определяется посредством последовательных приближений, если не поставлен факторный эксперимент, учитывающий взаимодействия. Из экономических соображений этот эксперимент следует проводить над очень малым количеством программ (кандидатов на изменение).

Разработать модель так, чтобы облегчить локальные изменения, конечно, легче, чем получить такую же гибкость по отношению к глобальным изменениям. Модульная структура с тщательным выбором и четким определением взаимодействия между модулями позволяет при большинстве локальных изменений ограничиться заменой или модификацией только одного модуля и лишь иногда требует внесения небольших изменений в общую базу данных модели.

Более легким и быстрым способом изменения модели для калибровки оказывается изменение значений калибровочных параметров. Предназначенные для этого параметры следует выбрать на этапе конструирования модели. Кроме того, их следует сделать легко доступными, т. е. включить в набор входных переменных модели. Часто калибровочные параметры принимают форму корректирующих коэффициентов в аналитических выражениях, используемых моделирующей программой. Например, в нашей модели мы вводим коэффициент k в формуле для вычисления $t_{дс}$ в табл. 3.5 следующим образом:

$$t_{дс} = k\bar{t}_{ддс} + t_{сдс}r_{дс}. \qquad (3.19)$$

Калибровочный параметр k можно интерпретировать как корректирующий множитель, предназначенный для частичной компенсации ошибок, возникающих из-за использования среднего $\bar{t}_{ддс}$ при вычислении $t_{дс}$, если распределение $t_{ддс}$ (как мы и ожидаем) заметно асимметрично. В этом случае распределение было бы не симметрично относительно среднего и наиболее вероятные значения $t_{ддс}$ существенно отличались бы от $\bar{t}_{ддс}$. Значение k должно быть определено в процессе калибровки. ■

Изменение калибровочных параметров — простейший и наиболее дешевый способ улучшения локальной степени детальности модели. Казалось бы структурные изменения, как более трудно осуществимые и дорогие, должны рассматриваться в последнюю очередь только после того, как все попытки откалибровать модель путем изменения параметров и локальных модификаций окажутся безуспешными. Это, однако, очень опасная стратегия. В действительности она может скрыть структурное несоответствие или недостаточную степень детальности, которые могут вызвать резкое сокращение области пригодности модели, не осознанное специалистами по моделированию. Противоположная стратегия, отдающая предпочтение глобальным модификациям по сравнению с локальными модификациями и с изменением калибровочных параметров, определенно безопаснее. Эта стратегия принята также в статистическом методе калибровки, описанном в следующем разделе.

3.3.3. Методика калибровки

Итеративная процедура калибровки, описанная в этом разделе, подобна предложенной Бейлнером и Вальдбаумом (1972) и представлена на рис. 3.17. Каждый из трех шагов на рис. 3.17 является по своей природе итеративным. Он включает ряд изменений модели, за-

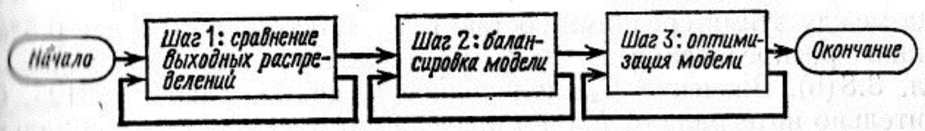


Рис. 3.17. Блок-схема метода калибровки Бейлнера и Вальдбаума.

каждым из которых следует тест для определения, завершен ли этот шаг или его следует продолжить. Методика является статистической, но применяется не только к стохастическим моделям. Действительно, Бейлнер и Вальдбаум использовали ее для калибровки полностью детерминированной модели, управляемой трассировочной записью и названной SOUL. Как будет показано ниже, единственное требование состоит в том, чтобы точность модели была определена как функция переменных, которые при модификациях модели ведут себя подобно случайным переменным. В нашем изложении мы предположим для простоты, что для описания производительности системы был выбран единственный индекс. Чтобы удовлетворять приведенному выше тре-

бованию, такой индекс должен отражать поведение системы при обработке каждого отдельного задания. Индексом такого типа является, например, время прохождения задания t.

Шаг. 1. Сравнение выходных распределений. Определением ошибки E, которая должна быть минимизирована в процессе калибровки, может служить разность между средними временами прохождения задания через систему и модель. Предполагая, что наше сравнение основано на обработке системой набора из N заданий и на обработке набора из N их моделей, получим

$$E_1 = \frac{1}{N}\sum_{j=1}^{N}(t_j - t'_j), \tag{3.20}$$

где, как указано в табл. 3.7, t_j есть время прохождения j-го задания в системе, а t'_j — время прохождения его модели через модель системы.

Более удовлетворительное определение ошибки относится к распределениям t и t' на рассматриваемых N заданиях. Пусть $F_t(\tau)$ и $F_{t'}(\tau)$ — функции распределения времен прохождения задания через систему и ее модель соответственно. Вспомним, что $F_t(\tau) = \text{Prob}(t \leqslant \tau)$ и $F_{t'}(\tau) = \text{Prob}(t' \leqslant \tau)$ при $0 \leqslant \tau < +\infty$. Тогда мы определим

$$E_2 = \int_0^{+\infty} |F_t(\tau) - F_{t'}(\tau)|\, d\tau. \tag{3.21}$$

Оценка E_2 лучше, чем E_1, так как она сравнивает локально, поточечно эти две функции распределения и суммирует абсолютные значения их разностей, чтобы избежать частичной компенсации ошибок из-за различия знаков. Рассмотрим, например, времена прохождения заданий в табл. 3.8(a). Их средние равны $\bar{t} = 14.15$ с и $\bar{t}' = 14.14375$ с. Разница между этими средними составит $E_1 = 0.00625$, т. е. менее 0.5% $\bar{t}$. Однако распределения t_j и t'_j различаются, как это видно из табл. 3.8(b). Значение E_2, полученное из (3.21), равно 0.13125. Относительно интеграла от $F_t(\tau)$ в пределах от 0 до 25.2 с (максимальное значение t_j по всем j) E_2 мало, хотя и много больше, чем E_1. Так как этот интеграл равен 12.3, то E_2 лишь немного превышает 1% от него.

Таким образом, даже критерий калибровки, основанный на E_2, можно интерпретировать как приводящий к выводу, что калибровка нашей модели окончена. Однако простого просмотра данных, приведенных в табл. 3.8(a) (который в нашем случае возможен, так как рассматривается только 16 заданий), достаточно, чтобы убедиться, что требуется дальнейшая калибровка. Действительно, для некоторых заданий разность между действительным и моделированным временем прохождения достигает 20%.

Рассмотрение значения E_1 или E_2 или просто визуальное сравнение графиков данных двух распределений часто наводит на мысль, как улучшить модель. Они также могут помочь нам определить, является ли некоторая произведенная модификация правильным улучшением

Таблица 3.8

Времена прохождения заданий, полученные системой и моделью

(a) Времена прохождения			(b) Функции распределения времен прохождения		
Номер задания j	t_j [с]	t'_j [с]	τ[с]	$F_t(\tau)$	$F_{t'}(\tau)$
1	12.2	14.7	4	0.0	0.0
2	8.1	7.9	5	0.0625	0.0625
3	10.2	12.3	8	0.0625	0.125
4	14.9	13.4	9	0.125	0.125
5	25.2	25.5	10	0.1875	0.1875
6	12.9	11.3	11	0.25	0.25
7	18.3	18.2	12	0.3125	0.3125
8	9.5	10.2	13	0.4375	0.375
9	13.3	13.9	14	0.5	0.5625
10	21.0	20.8	15	0.6875	0.6875
11	16.2	16.1	17	0.75	0.75
12	14.6	13.1	19	0.8125	0.8125
13	11.2	9.4	20	0.8125	0.875
14	14.1	14.9	21	0.9325	0.9325
15	20.1	19.9	26	1.0	1.0
16	4.6	4.7			

или нет. Этот шаг в основном предполагает макроскопические изменения, например модификации структуры или замены главных программ.

Читателю следует помнить, что получение результатов моделирования, распределения которых идентичны или очень близки к реальным, является необходимым, но не достаточным условием для того, чтобы модель была точна в разумно большой области. Вполне допустимо, что способ обработки моделью некоторого задания существенно отличается от способа обработки того же задания в моделируемой системе, даже если модель демонстрирует при одном или нескольких входных условиях те же выходные распределения, что и система. Таким образом, использование E_1 или E_2 в качестве критериев калибровки не гарантирует правильной работы модели при входных условиях, не рассмотренных в процессе калибровки.

Шаг. 2. Балансировка модели. Более удобным, чем E_1 и E_2, критерием калибровки для сравнения реальной системы с моделью по времени прохождения задания является критерий, относящийся к распределению разности времен для каждого задания. Возможным критерием

такого типа является среднее квадратов разностей:

$$E_3 = \frac{1}{N}\sum_{j=1}^{N}(t_j - t'_j)^2. \qquad (3.22)$$

Например, значение E_3 для результатов в табл. 3.8 (*а*) равно 1.418. Его квадратный корень составляет 8.4% $\bar{t}$. Таким образом, этот критерий калибровки дает результат, лучше согласующийся с тем, что мы видим в табл. 3.8(а), чем критерии, основанные на E_1 или E_2.

По мере усложнения процедуры калибровки и по мере удаления все более макроскопических отклонений трудность определения необходимых изменений возрастает. Сравнение производительности по каждому заданию, подобное лежащему в основе E_3, может обнаружить, например, что модель обрабатывает некоторые задания быстрее, а некоторые медленнее, чем система. Но на какие части модели можно наложить ответственность за эти искажения?

Чтобы ответить на этот вопрос, мы должны проанализировать зависимость E_3 от различных характеристик заданий. Например, обнаружив, что модель ускоряет выполнение заданий с большим запросом на память и замедляет с малым, мы сразу бы сделали вывод, что надо исследовать механизм планирования ЦП и стратегию распределения памяти в модели.

Определения ошибок, подобные E_1, и E_3, демонстрируют вариации, которые при сопоставлении их с параметрами заданий выглядят как случайные, даже если модель полностью детерминированная. Обычно эти вариации можно разложить на функциональный компонент *(тренд)* и случайный компонент *(шум)*. Таким образом, статистический метод типа регрессионного анализа (см. разд. 2.7.3), по-видимому, применим к изучению чувствительности E_3 к изменению параметров задания. В случае модели, разработанной в примерах 3.2 и 3.4, параметры в табл. 3.6, кроме $t_{\text{приб}}$, параметров $n_{\text{цп}}$ и B, можно рассматривать как независимые переменные в уравнении регрессии для E_3. Данные, необходимые для вычисления коэффициентов регрессии, можно получить за один прогон модели.

Затем тесты значимости, описанные в разд. 2.7.3, выполняются для каждого коэффициента (частичные F-тесты) и для уравнения регрессии (F-тест). Если уравнение регрессии окажется значимым на выбранном доверительном уровне, параметры задания, коэффициенты которых значимы, прямо указывают на изменения, которые следует сделать для уменьшения значимости коэффициентов. Заметим, что эти изменения могут оказаться глобального типа, но в большинстве случаев можно обойтись локальными изменениями. Целью этой процедуры, которая называется *методом хорошего баланса*, является получение незначимой регрессии. Это соответствует ситуации, в которой разность между выходами модели и системы не зависит существенно от характеристик задания. При таком *хорошо сбалансированном* моделировании, модель системы обрабатывает все задания точно так же, как и сама

система. Для достижения этой точки регрессионный анализ должен обеспечивать только индикацию, а линейной формы регрессионного уравнения, по-видимому, достаточно. Заметим, однако, что выбор уровней значимости должен делаться очень осторожно, так как предположения, при которых F-тест справедлив (см. разд. 2.7.2), выполняются не всегда.

Шаг 3. Оптимизация модели. В хорошо сбалансированной модели значение E_3 не обязательно равно нулю или очень мало. Последний шаг процедуры калибровки использует методы статистической оптимизации для минимизации E_3. Подход, примененный Бейлнером и Вальдбаумом (1972), является статистической версией метода наискорейшего спуска.

На этом шаге рассматриваются только изменения калибровочных параметров. Выполняется несколько прогонов модели системы при различных значениях калибровочных параметров. Пусть $\mathbf{H}$ — вектор калибровочных параметров, а $\mathbf{H}_b$ — «наилучшее» значение $\mathbf{H}$ среди тех, которые использовались при различных прогонах модели системы. Иначе говоря, $\mathbf{H}_b$ есть вектор, дающий минимальное значение E_3. Выполняется линейная регрессия между калибровочными параметрами и E_3, основанная только на данных, собранных во время прогонов, соответствующих значениям $\mathbf{H}$, достаточно близким к $\mathbf{H}_b$. Например, мы можем рассматривать только точки внутри гиперкуба с центром в $\mathbf{H}_b$ в пространстве калибровочных параметров. Если регрессия значима, уравнение регрессии, содержащее только те члены, коэффициенты которых значимы, может рассматриваться как локальная линейная аппроксимация функции $E_3(\mathbf{H})$ после уничтожения шумового компонента. Направление вектора параметров регрессии $\mathbf{B}$ близко к направлению наискорейшего спуска этой функции. Таким образом, следующим «хорошим» значением $\mathbf{H}$ окажется

$$\mathbf{H}_{bn} = \mathbf{H}_b - c\mathbf{B} \quad (c > 0). \qquad (3.23)$$

Прогон модели выполняется с $\mathbf{H} = \mathbf{H}_{bn}$. Полученное значение E_3 сравнивают с полученным при $\mathbf{H}_b$. Так как E_3 ведет себя аналогично случайной величине, то сравнения двух значений ее, вообще говоря, не достаточно для определения, является ли модифицированная модель значимо лучшей, чем модель с $\mathbf{H} = \mathbf{H}_b$. Поэтому соответствующие статистические тесты [(см. например, Снедекор и Кокрен (1967)] должны основываться на последовательности пар наблюдений, состоящих из квадратов разностей $(t_j - t'_j)$ для каждого задания с номером j. Если эти тесты покажут, что модифицированная модель системы значимо лучше, чем предыдущая, выполняется новая локальная регрессия вблизи $\mathbf{H}_{bn}$ и вычисляется следующее значение $\mathbf{H}$ с использованием (3.23). В противном случае этот шаг и вся процедура калибровки заканчиваются.

Скорость сходимости итеративных методов, использованных в трех шагах данной процедуры, сильно зависит от того, насколько хорошо сконструирована модель системы. Она также, по-видимому,

зависит от выбора вносимых изменений и их масштаба. Поскольку каждая итерация обычно обходится довольно дорого, очень важна скорость сходимости. Бейлнер и Вальдбаум сумели откалибровать модель системы SOUL, использовав очень немного итераций на шаг. Важным требованием к модели калибруемой системы является очень низкое отношение реального времени к времени модели, так чтобы каждый прогон был относительно быстрым и недорогим. Подход к разработке модели системы, не противоречащий этому требованию, — это подход, начинающийся с разработки грубого каркаса модели и использующий процедуру калибровки для последовательного ее совершенствования, пока она не достигнет желаемой точности.

3.4. ПЛАНИРОВАНИЕ ИМИТАЦИОННЫХ ЭКСПЕРИМЕНТОВ

3.4.1. Планирование эксперимента при моделировании

Проблемы, встречающиеся при планировании имитационного эксперимента, концентуально очень похожи на проблемы, рассмотренные в гл. 2 для измерительных экспериментов. Это сходство часто позволяет нам подходить к ним с одинаковыми методами. Например, то, что было сказано в разд. 2.6.2 и 2.6.3 о применении статистических методов планирования эксперимента к измерительным работам, можно повторить для работ по имитационному моделированию.

Входные переменные модели часто определяют как стохастические, чтобы эффективно воспроизвести изменчивость соответствующих параметров установки. Все переменные модели наблюдаемы, и все входные переменные поддаются управлению, даже если они стохастические. Разумеется, параметры установки, не представленные в модели, из-за ограничений, наложенных создателями модели на ее степень детальности, не могут быть выбраны в качестве факторов или индексов ни в каких экспериментах с этой моделью. Индексы производительности и факторы должны выбираться при создании модели. Так как прогоны модели, вообще говоря, дороги, количество факторов и число уровней каждого из них должны быть минимальными, совместимыми с целями эксперимента.

В разд. 2.6.2 мы видели, что вторичные факторы, характеризующие рабочую нагрузку, иногда приходится вводить в измерительные эксперименты, в которых система управляется своей производственной рабочей нагрузкой. Эти факторы можно наблюдать, но ими нельзя эффективно управлять. Таким образом, как описано в разд. 2.6.3, их эффекты должны объединяться в блоки посредством подходящего планирования эксперимента. Эта проблема не возникает в имитационных экспериментах. В модели рабочая нагрузка представлена входными переменными. Следовательно, она всегда управляема и точно воспроизводима.

Имитационные эксперименты характеризуются намного более легкой воспроизводимостью, чем измерительные. Легко инициализиро-

Таблица 3.9

Факторы и уровни для эксперимента на модели из примера 3.6

Факторы		Уровни		
Имя	Символ	Имя	Символ	Описание
Размер памяти	M	МАЛЫЙ	$M1$	$32K$ слов
		СРЕДНИЙ	$M2$	$64K$ слов
		БОЛЬШОЙ	$M3$	$96K$ слов
Скорость барабана	R	МЕДЛЕННАЯ	$R1$	$\overline{t}_{\text{дбр}}=8$ мс, $\overline{t}_{\text{сбр}}=7$ мкс
		БЫСТРАЯ	$R2$	$\overline{t}_{\text{дбр}}=5$ мс, $\overline{t}_{\text{сбр}}=4$ мкс
Скорость диска	S	МЕДЛЕННАЯ	$S1$	$\overline{t}_{\text{ддс}}=50$ мс, $\overline{t}_{\text{сдс}}=12$ мкс
		БЫСТРАЯ	$S2$	$\overline{t}_{\text{ддс}}=30$ мс, $\overline{t}_{\text{сдс}}=8$ мкс

вать модель в точно том же состоянии и управлять ею в точности тем же входным потоком столько прогонов, сколько потребуется. Когда выбраны уровни всех факторов, экспериментаторам в идеале следует попытаться проверить правильность модели для всех условий, соответствующих комбинациям всех уровней.

Пример 3.6. Планируемый эксперимент, который должен быть выполнен на модели, описанной в примерах 3.2 и 3.4, представлен в табл. 3.9 и 3.10. Работа, в которую входит этот эксперимент, направлена на создание установки пакетной обработки. Модель предназначена для получения информации об ожидаемой производительности установки. Она должна помочь сделать выбор из нескольких вариантов размера основной памяти, скоростей барабана и диска. Рассматриваются три размера основной памяти, два типа дисков и два барабанов. Факторный план, требующий 12 прогонов, представлен в табл. 3.10. В каждом прогоне с $M=Mi$, $R=Rj$ и $S=Sk$ измеряются средняя пропускная способность $\overline{T}_{ijk}$ и среднее время прохождения задания $\overline{t}_{ijk}$. На основе 12 пар значений этих двух индексов производительности будет сделано техническое обоснование для выбора диска, барабана и основной памяти. ■

Литература по моделированию вычислительных систем содержит большое число статей, описывающих эксперименты с многократными прогонами, выполненные для различных целей, таких, как выбор

Таблица 3.10

План факторного эксперимента на модели из примера 3.2

		R, S			
		$R1$, $S1$	$R1$, $S2$	$R2$, $S1$	$R2$, $S2$
M	$M1$	$\overline{T}_{111}$, $\overline{t}_{111}$	$\overline{T}_{112}$, $\overline{t}_{112}$	$\overline{T}_{121}$, $\overline{t}_{121}$	$\overline{T}_{122}$, $\overline{t}_{122}$
	$M2$	$\overline{T}_{211}$, $\overline{t}_{211}$	$\overline{T}_{212}$, $\overline{t}_{212}$	$\overline{T}_{221}$, $\overline{t}_{221}$	$\overline{T}_{222}$, $\overline{t}_{222}$
	$M3$	$\overline{T}_{311}$, $\overline{t}_{311}$	$\overline{T}_{312}$, $\overline{t}_{312}$	$\overline{T}_{321}$, $\overline{t}_{321}$	$\overline{T}_{322}$, $\overline{t}_{322}$

компьютера и разработка конфигурации [см., например, Нильсен (1967)], разработка системы [см., например, Нильсен (1971) и Шерман и др. (1972)], а также повышение производительности [см., например, Нильсен (1967)]. В статье Гоела и Лиу (1973) описан статистически спланированный имитационный эксперимент.

3.4.2. Продолжительность прогона модели

Задача определения продолжительности прогона модели концептуально аналогична задаче выбора продолжительности сеанса измерения. Таким образом, то, что было сказано в разд. 2.6.4 об этой задаче и ее решении для измерительных экспериментов, также справедливо для имитационных экспериментов. Мы можем сформулировать основной тезис разд. 2.6.4 следующим образом. Часто наши наблюдения поведения модели нельзя рассматривать как независимые и одинаково распределенные. Следовательно, мы не можем применить центральную предельную теорему, как мы это делали в разд. 2.4.3, для оценки дисперсий статистических показателей на основе наших наблюдений. Эти дисперсии относительно неизвестных истинных средних нужны для вычисления точности оценок наших индексов производительности. Как только все оценки достигают желаемой точности, прогон можно остановить. Методы оценки дисперсии, введенные в разд. 2.6.4 (методы автоковариаций, независимых копий и метод подвыборок), можно применить к имитационному эксперименту. Фактически, они применялись и продолжают применяться гораздо шире в моделировании, чем в измерительных экспериментах. Оценка дисперсии в реальном масштабе времени, требовавшая в измерительных экспериментах достаточно быстрого и мощного внешнего измерителя с запоминаемой программой, легче выполняется при прогонах модели. Мы видели в примере 3.2, что избежать искажений во времени и пространстве модели легко, даже если моделирующая программа использует реальные ресурсы.

Дисперсию статистической оценки не только легче оценить при прогоне модели, чем во время сеанса измерения, но и ее также можно уменьшить, применяя специальные *методы уменьшения дисперсии*. Наиболее распространенные из них оперируют потоками псевдослучайных чисел, поступающих на вход модели. Таким образом, они применимы только к стохастическому моделированию.

Как упоминается также в разд. 5.3.5, в измерительных экспериментах могут использоваться искусственные рабочие нагрузки, которые обладают характеристиками моделей рабочей нагрузки, удобными для моделей, управляемых распределениями. Было сделано предположение [см. Феррари и др. (1975)], что если система управляется искусственными рабочими нагрузками этого типа, эти методы уменьшения дисперсии можно эффективно применить для сокращения длительности сеанса измерения или для повышения точности статистических оценок.

Стохастическая модель получает одну или несколько последовательностей псевдослучайных чисел во время каждого из своих прогонов. Различные последовательности можно получать от разных генераторов или от одного генератора, инициализированного при различных начальных значениях. Для упрощения наших рассмотрений сосредоточим наше внимание на одной последовательности, которая является либо единственной, либо одной из используемых моделирующей программой при определенном прогоне. Обозначим последовательность через $Z=z_1, z_2, \ldots$ ($0 \leqslant z_i < 1$ для всех $i=1, 2, \ldots$), и пусть z_0 — ее начальное значение.

Во время прогона мы собираем наблюдения величины u. Эти наблюдения будут использованы для оценки индекса производительности. Например, значениями u могут быть времена прохождения, времена ответа, длины очередей. Обозначим последовательность $u_1, u_2, \ldots$ через $U(Z)$, чтобы подчеркнуть тот факт, что последовательность получена, когда модель управлялась (среди всего прочего) потоком псевдослучайных чисел Z.

Среднее $\bar{u}_n$ по первым n наблюдениям $U(Z)$ является оценкой истинного среднего $\bar{u}$, индекса производительности, который мы хотим определить посредством моделирования. Если мы инициализируем генератор псевдослучайных чисел при другом начальном значении z'_0 для всех $z_i \in Z$, то получим новый поток Z', а модель даст новый ряд наблюдений $U'(Z')=u_1', u'_2, \ldots$. Среднее $\bar{u}'_n$ первых n наблюдений из U' является другой оценкой неизвестного истинного среднего $\bar{u}$.

Если $\bar{u}_n$ и $\bar{u}'_n$ — несмещенные оценки $\bar{u}$, мы можем рассматривать их как случайные переменные, имеющие одинаковое среднее $\bar{u}$. Таким образом, случайная переменная

$$\bar{u}^*_{2n} = a\bar{u}_n + (1-a)\,\bar{u}'_n \quad (0 < a < 1) \tag{3.24}$$

обладает средним $\bar{u}$ и дисперсией

$$\mathrm{Var}(\bar{u}^*_{2n}) = a^2\,\mathrm{Var}(\bar{u}_n) + (1-a)^2\,\mathrm{Var}(\bar{u}'_n) + 2a(1-a)\,\mathrm{Cov}(\bar{u}_n, \bar{u}'_n). \tag{3.25}$$

Заметим, что $\bar{u}^*_{2n}$ — оценка $\bar{u}$ по $2n$ образцам. В общем с ростом n мы можем ожидать меньшего уменьшения дисперсии $\bar{u}_n$, чем в случае независимых наблюдений (см. разд. 2.6.4). Таким образом,

$$\mathrm{Var}(\bar{u}_{2n}) \geqslant \frac{\sigma^2}{2n}, \tag{3.26}$$

где σ^2 — дисперсия случайных переменных $u_1, u_2, \ldots$, которые предполагаются одинаково распределенными. По этой же причине мы также находим, что

$$\mathrm{Var}(\bar{u}_n) \geqslant \frac{\sigma^2}{n}, \quad \mathrm{Var}(\bar{u}'_n) \geqslant \frac{\sigma^2}{n}. \tag{3.27}$$

Для простоты предположим, что левые части неравенств (3.27) равны произведениям k_n на соответствующие правые части, где $k_n \geqslant 1$.

Аналогично, (3.26) можно написать в виде равенства, вводящего коэффициент $k_{2n} \geqslant 1$. Подставляя в (3.25), получим

$$\mathrm{Var}\,(\bar{u}^*_{2n}) = \frac{2k_n}{k_{2n}}(2a^2 - 2a + 1)\,\mathrm{Var}\,(\bar{u}_{2n}) + 2a\,(1-a)\,\mathrm{Cov}\,(\bar{u}_n, \bar{u}'_n). \qquad (3.28)$$

Это уравнение показывает, что при определенных условиях дисперсия оценки, основанной на $2n$ наблюдениях, такой, как $\bar{u}_{2n}$, может быть уменьшена применением статистических оценок $\bar{u}_n$ и $\bar{u}'_n$, каждая из которых основана на n наблюдениях. Если сделанные выше предположения выполнены, условие уменьшения дисперсии ($\mathrm{Var}\,(\bar{u}^*_{2n}) < \mathrm{Var}\,(\bar{u}_{2n})$) есть

$$\mathrm{Cov}\,(\bar{u}_n, \bar{u}'_n) < \frac{1 - 2k_n\,(2a^2 - 2a + 1)/k_{2n}}{2a\,(1-a)}\,\mathrm{Var}\,(\bar{u}_{2n}). \qquad (3.29)$$

Для $k_{2n} = k_n$ и $a = 1/2$ условие (3.29) сводится к

$$\mathrm{Cov}\,(\bar{u}_n, \bar{u}'_n) < 0. \qquad (3.30)$$

Можно попытаться получить отрицательно коррелированные оценки, применив *метод антитетических переменных*. После прогона модели с *правильным потоком* Z и вычисления $\bar{u}_n$ мы осуществляем прогон модели с *антитетическим потоком* $Z' = 1 - Z = 1 - z_1, 1 - z_2, \ldots$, и вычисляем $\bar{u}'_n$. Оценка $\bar{u}^*_{2n}$ индекса производительности $\bar{u}$ дается выражением (3.24). Разумеется, этот метод применим к оценке любого числа индексов производительности.

Метод антитетических переменных основан на надежде, что отрицательная корреляция, созданная между входными воздействиями, будет каким-то образом распространяться к выходу модели. Более вероятно, что это произойдет, если порядок событий внутри модели существенно не отличается при различных входных потоках и если пределы выходных сигналов некоторым образом монотонно связаны со значениями псевдослучайных чисел входного потока. Например, если эти условия выполнены, время прохождения задания, для которого время ЦП определяется малым случайным числом z_i, будет, вероятно, короче, чем для задания с временем ЦП, вычисленным по $1 - z_i$. К сожалению, нет оснований думать, что эти условия выполняются в многопрограммных вычислительных системах. Напротив, интуитивно ясно, что они вряд ли справедливы.

Так как из (3.28) видно, что $\mathrm{Var}\,(\bar{u}^*_{2n})$ может быть даже больше, чем $\mathrm{Var}\,(\bar{u}_{2n})$, то мы всегда можем оценить обе дисперсии и выбрать оценку с меньшей. Однако, если оценка $\bar{u}_{2n}$ существенно лучше, чем $\bar{u}^*_{2n}$, дополнительные накладные расходы, обусловленные использованием антитетического потока, более не компенсируются никакими преимуществами, и следует или избегать этого метода, или пытаться модифицировать, если возможно, модель системы.

Те же самые замечания можно повторить для *метода стратифицированных выборок*, у которого стратегия получения отрицательных корреляций между выходными переменными состоит в получении

$(Z + d) \bmod 1$, где $0 < d < 1$. Описания и оценки других методов уменьшения дисперсии можно найти в литературе по моделированию [см., например, Гордон (1969), Нейлор (1971), Фишман (1973)].

В планировании имитационных экспериментов, так же как и в некоторых измерительных экспериментах с использованием искусствен-

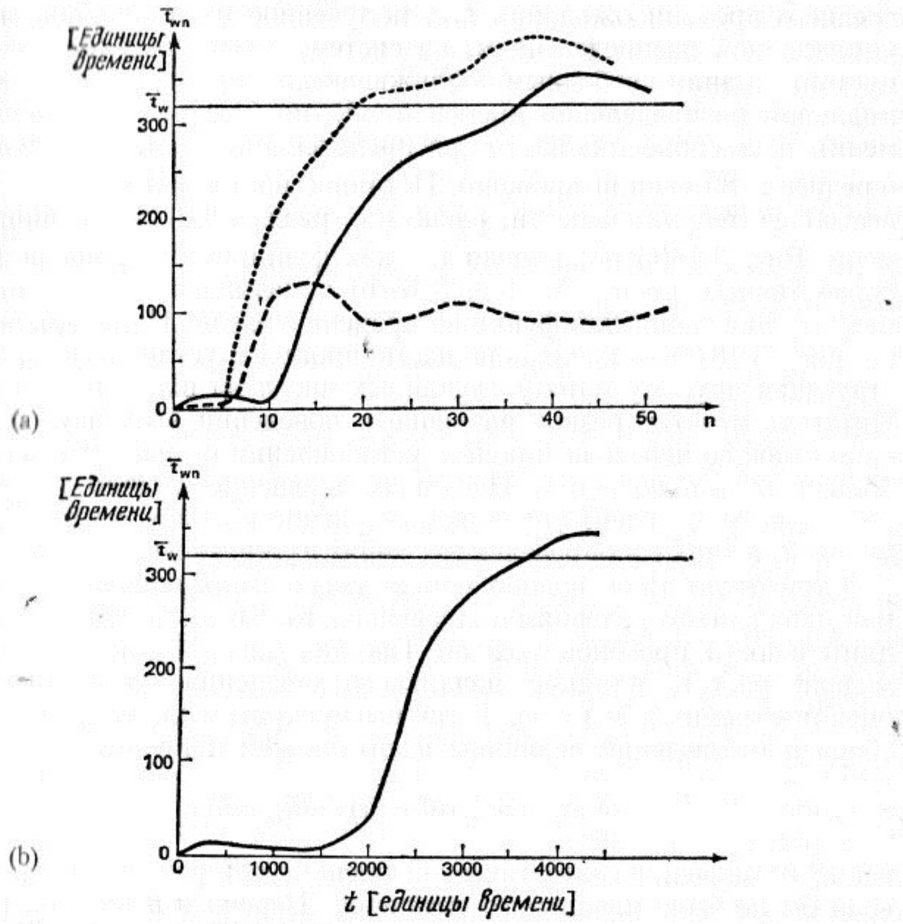


Рис. 3.18. Начальные этапы процессов установления при моделировании системы с очередями (экспоненциально распределенное время между поступлениями, экспоненциально распределенное время обслуживания, обслуживание поодиночке). Индекс производительности: среднее время ожидания $\bar{t}_w$ для задания изображено как функция (a) числа обработанных заданий и (b) времени модели от начала прогона. [*Источник*: Рехтшаффен (1972).]

ной рабочей нагрузки, всегда присутствует проблема *начальных условий*. Мы обычно интересуемся распределениями стационарных состояний или средними значениями индексов производительности. Однако наши наблюдения вообще подвержены воздействию начального перехода, характеристики которого зависят от начального состояния модели (или, в измерительном эксперименте, от начального состояния системы). До сих пор в нашем обсуждении для простоты мы полагали,

что выборочные средние значения не смещены оценками средних значений индексов производительности. В действительности эти оценки часто смещены из-за начального процесса установления.

Рис. 3.18, основанный на данных, опубликованных Рехтшаффеном (1972), показывает самое начало процесса установления выборочного среднего времени ожидания $\bar{t}_{wn}$, полученное имитационной моделью, управляемой распределением, на системе с очередью (первоначально пустой) с одним свободным обслуживающим прибором, при экспоненциальном распределении времен прибытия (среднее: 100 единиц времени) и экспоненциальным распределением времен обслуживания (среднее : 80 единиц времени). Истинное время ожидания $\bar{t}_w$, как известно из теории очередей, равно (см. разд. 4.3.2) 320 единицам времени. Рис. 3.18(a) показывает $\bar{t}_{wn}$ как функцию от n, числа заданий, обработанных системой. Рис. 3.18(b) показывает тот же самый процесс установления как функцию времени модели. Пунктирная кривая на рис. 3.18(a) — начальная часть процесса установления, соответствующая другому потоку случайных чисел на входе той же модели. Читатель отметит резкое различие в поведении этих двух кривых в самом начале перехода (процесс установления обычно занимает много больше 50 наблюдений). Начальное смещение обычно присутствует не только у моделей, управляемых распределением, но во всех типах моделей, а также в определенных типах измерительных экспериментов.

Существует три основных метода уменьшения ошибки, обусловленной начальными условиями. Первый состоит в достаточном увеличении длительности прогонов модели. Так как число наблюдений и время модели растут, влияние начального смещения на статистическую оценку стремится к нулю. Если мы прекратим моделирование после сбора n наблюдений величины u, то сможем написать

$$\bar{u}_n = \bar{u} + (u_n^0 - \bar{u}) + (\bar{u}_n - \bar{u}_n^0), \tag{3.31}$$

где $\bar{u}_n^0$ — значение статистической оценки, которое мы бы вычислили, если бы не было начального смещения. Перенося $\bar{u}$ (символ неизвестного истинного среднего) в левую часть (3.31), возводя в квадрат и взяв математическое ожидание от обеих частей, мы получим среднеквадратичную ошибку оценки $\bar{u}_n$:

$$MSE = E[(\bar{u}_n - \bar{u})^2] = E[(\bar{u}_n^0 - \bar{u}_n)^2] + E[(\bar{u}_n - \bar{u}_n^0)^2]. \tag{3.32}$$

Заметим, что член $E[2(u_n^0 - \bar{u})(\bar{u}_n - \bar{u}_n^0)]$ не появляется в правой части (3.32), так как мы предполагаем, что две случайные переменные $(u_n^0 - \bar{u})$ и $(u_n - \bar{u}_n^0)$ независимы и, следовательно, их ковариация равна нулю. Таким образом, среднеквадратичная ошибка оценки $\bar{u}_n$ состоит из двух компонентов. Первый представляет собой ее дисперсию относительно истинного среднего $\bar{u}$ и часто зависит от n как $1/n$. Второй есть средний квадрат смещения, и, так как смещение обычно уменьшается как $1/n$ с ростом n, его можно рассматривать как член порядка

$1/n^2$. Это означает, что влияние смещения убывает быстрее, чем влияние дисперсии, с ростом продолжительности прогона модели.

Какова минимальная длительность, гарантирующая эффект смещения, меньший заданного значения? Чтобы ответить на этот вопрос, мы должны были бы знать зависимость смещения от числа наблюдений. Ощущение скорости убывания начального смещения можно получить во время прогона, подсчитывая число пересечений выборочного среднего в последовательности наблюдений $u_1, u_2, \ldots$. Это будет количество случаев, когда $u_i < \bar{u}_i$ и $\bar{u}_{i+1} > \bar{u}_i$ или $u_i > \bar{u}_i$ и $u_{i+1} < \bar{u}_i$. Чем больше это число, тем быстрее уменьшаются при моделировании эффекты начальных условий [см. Фишман (1973)].

Второй метод уменьшения ошибки, обусловленной начальным смещением, состоит в инициализации модели в специально выбранном состоянии. Это состояние могло бы быть одним из тех, в которых модель оказывается в конце предыдущего прогона (или где-нибудь в середине его). Этот метод требует запоминания подходящего состояния или информации, достаточной для его восстановления, во время прогона и наличия программы инициализации (см. рис. 3.9), воспроизводящей его в начале следующего прогона.

Третий метод состоит в том, чтобы начинать сбор наблюдений после определенного интервала времени модели или после поступления определенного числа заданий, вместо того чтобы начинать сбор наблюдений с начала прогона. Динамическое поведение такой статистической оценки, как выборочное среднее, во время начального перехода может быть весьма чувствительно к выбору точки начала сбора данных.

Например, точечная кривая на рис. 3.18 (a) представляет среднее время ожидания, когда первые пять наблюдений исключены. Заметим, что из этого рисунка, показывающего как раз самое начало перехода, никакого другого заключения сделать нельзя. Если бы наблюдения проводились, начиная с поступления десятого задания или позже, кривая $\bar{t}_{wn}$ этого примера пошла бы вверх очень быстро и определенно осталась бы выше $\bar{t}_w$ в течение всего интервала, показанного на рис. 3.18.

Обычно число измерений n_0, которые следует исключить, намного больше 5 и даже 50. Возможный метод определения его состоит в прогоне модели для нескольких коротких периодов времени с различными входными воздействиями и одним и тем же начальным состоянием и в последующем построении графиков наблюдений. Затем можно выбрать в качестве n_0 первое значение n, в котором ни один из временных рядов, изображенных на графике, не имеет ни максимума, ни минимума [см. Конвей (1963)].

Исключение первых n_0 наблюдений (или, что то же самое, инициализация модели в непустом состоянии) уменьшает начальное смещение, но обычно увеличивает дисперсионный член в (3.32). Таким образом, экспериментатор часто оказывается перед проблемой выбора: исключать или не исключать? Нежелательные влияния первого решения на дисперсию могут потребовать сделать прогоны длиннее, чем

это было нужно для получения той же точности без попыток уменьшения смещения. С другой стороны, последнее решение (не исключать) в некоторых случаях хуже из-за особенно большого начального смещения.

Если для целей оценки дисперсии мы используем метод независимых копий, каждый прогон заменяется на h независимых подпрогонов, каждый из которых имеет свой начальный процесс установления. Таким образом, продолжительность каждого подпрогона должна быть настолько велика, чтобы ошибка за счет начального смещения была пренебрежимо мала, и в то же время h должно быть велико, чтобы уменьшить дисперсионный компонент ошибки. В методе подвыборок мы имеем только один начальный процесс установления на прогон. Однако число h подвыборок должно быть большим, чтобы уменьшить дисперсионный компонент, а число q наблюдений в подвыборке должно быть большим, чтобы сделать оценку дисперсии достаточно точной.

Читателю следует также отметить, что условия окончания прогона, которые Фишман (1973) называет *конечными условиями*, могут вносить смещение в некоторые статистические оценки. Это верно, например, для выборочного среднего длины очереди, если условием завершения является окончание n-го задания при данном n. Смещение обусловлено тем, что интервал времени модели, необходимый для обработки n заданий, есть случайная переменная, как и число выборок, периодически собираемых для определения средней длины очереди. Конечное смещение воздействует также на оценку средней пропускной способности в измерительных или имитационных экспериментах, состоящих во вводе в систему (или в модель системы) n заданий и в измерении общего затраченного времени до завершения последнего задания. Если смесь заданий содержит некоторые задания, времена выполнения которых намного больше, чем у остальных, то порядок, в котором рассматриваются n заданий, может оказать заметное влияние на общее затраченное время. Когда очень продолжительное задание обрабатывается последним, система (или ее модель) долгое время работает в однопрограммном режиме, практически без перекрытия между работой центрального процессора и каналов ввода-вывода, что оказывает очевидное и нежелательное воздействие на общее затраченное время. Это часто называют *хвостовым* или *концевым эффектом*. Чтобы уменьшить или ликвидировать его, порядок поступления заданий в систему и порядок их обработки должны быть тщательно изучены и в максимально возможной степени контролироваться. Прекращение эксперимента до завершения обслуживания всех заданий потребовало бы однозначного определения момента прекращения. Это крайне трудная задача, так как момент прекращения должен быть единым образом определен для любого порядка заданий и должен быть относительно независим от порядка поступления, хотя и достаточно чувствительным к изменениям системы, рассматриваемым в исследовании. Значение хвостового эффекта для проектирования выполняемой искусственной рабочей нагрузки обсуждается в конце разд. 5.3.5.

Обобщим теперь введенные в этом разделе понятия, продолжив обсуждение эксперимента, описанного в примере 3.6.

Пример 3.7. Спланировав эксперимент, который мы хотим выполнить, как описано в примере 3.6, мы должны теперь выбрать количество заданий, подлежащих обработке во время каждого из 12 прогонов. Нашими первичными индексами производительности, которые предстоит оценить, являются два средних $\bar{T}_{ijk}$ и $\bar{t}_{ijk}$. В принципе минимальное значение n, удовлетворяющее одним и тем же требованиям точности, может значительно меняться от прогона к прогону. Однако мы будем стремиться к определению значения n, приемлемого для всех прогонов. Это решение позволяет нам использовать для управления моделью системы одну и ту же модель рабочей нагрузки во всех прогонах.

Мы в первую очередь интересуемся значениями индексов производительности для каждой конфигурации, поскольку хотим рассматривать только те из 12 конфигураций, производительность которых, вероятно, окажется приемлемой для пользователей установки. Тогда для подходящих конфигураций важнее получить точные относительные значения этих индексов. Фактически наша цель — получить сравнительную картину различий производительности, соответствующих различным ценам конфигураций. Эти соображения помогут нам специфицировать необходимую точность оценки индексов производительности. Точность можно выразить в терминах доверительных интервалов и доверительных уровней, как показано в разд. 2.6.4.

Для вычисления минимального значения n, необходимого для достижения желательной точности, выбран метод оценки дисперсий статистических оценок $\bar{T}$ и $\bar{t}$. Мы выберем метод подвыборок (см. разд. 2.6.4). Каждой комбинации уровней в табл. 3.10 будет соответствовать относительно продолжительный прогон модели, который будет подразделен с точки зрения сбора данных на h подпрогонов.

Как отмечалось ранее в этом разделе, метод подвыборок привлекателен тем, что большая продолжительность прогона уменьшает важность проблемы начального смещения. Однако число наблюдений в каждой подвыборке имеет нижнюю границу, которая диктуется необходимостью точной оценки дисперсии (см. разд. 2.6.4). Для выбора подходящих значений для q и h мы используем описанный в примере 2.9 метод сбора значений $\bar{T}$ и $\bar{t}$ для относительно коротких подпрогонов в течение одного или двух рабочих прогонов и комбинирования их до тех пор, пока не будет получен удовлетворительный уровень точности. Так как мы не знаем, какая из 12 конфигураций дает наибольшую дисперсию каждой из статистических оценок, то наш рабочий прогон мог бы выполняться для конфигурации с промежуточной производительностью, например соответствующей уровням $M2$, $R2$, $S2$ (см. табл. 3.9). Затем минимальное значение n, полученное этим способом, можно увеличить, умножив на фактор безопасности.

Вместо этого мы могли бы провести два рабочих прогона — один с уровнями $M1$, $R1$, $S1$, а другой — с уровнями $M3$, $R2$, $S2$. Эти два прогона прозондировали бы производительности двух крайних конфигураций. Однако не существует доказательства, что дисперсия ведет себя подобно средним. Мы не знаем, будут ли минимальные значения n, соответствующие остальным 10 конфигурациям, ограничены этими двумя значениями n, полученными из исследовательских прогонов. Таким образом, в этом случае также следует ввести фактор безопасности.

Проблемы начального и конечного смещений должны рассматриваться до выполнения рабочих прогонов. Следует ожидать, что начальные условия будут воздействовать, в частности, на средние значения по первым подвыборкам. Чтобы ослабить их воздействие до допустимых пределов, исключим первые n_0 наблюдений каждого прогона. Различные прогоны могут иметь различные минимальные значения n_0 при заданной максимальной ошибке из-за начального смещения. Используя метод, предложенный ранее в этом разделе, мы можем определить n_0 для рабочего прогона (прогонов). Затем умножим его значение (или наибольшее из двух значений) на фактор безопасности и примем результат в качестве числа заданий, вкладом которых в $\bar{T}$ и $\bar{t}$ можно пренебречь во всех прогонах, включая рабочие. Если это число обозначено через n_0^*, в каждом прогоне будет исполняться фактически $n_0^* + hq$ заданий. Интервал

времени модели, на протяжении которого будет вычисляться средняя пропускная способность, заключен между поступлением (n_0^*+1)-го задания и уходом (n_0^*+hq)-го.

О проблеме хвостового эффекта следует заботиться при моделировании нормальной деятельности системы, включая поступление новых заданий и их обработку, вплоть до ухода (n_0^*+hq)-го задания. Это решение может обусловить некоторое различие моделей рабочей нагрузки для различных прогонов, даже если модель управляется трассировочной записью событий или программно. Действительно, из-за мультипрограммирования и из-за различия конфигураций, задание, которое должно быть завершено последним в одном прогоне, может быть лишь частично обработано или завершено раньше, чем последнее в другом прогоне. Однако это, вероятно, еще малое искажение, менее важное, чем то, которое будет вызвано хвостовым эффектом, если мы не моделируем поступления новых заданий после окончания (n_0^*+hq)-го.

Другим вопросом, требующим ответа при планировании эксперимента, будет вопрос, следует ли использовать какой-либо метод уменьшения дисперсии. Для принятия решения мы должны сравнить ожидаемые преимущества этого метода с ожидаемыми затратами. Предполагая, что модель управляется распределением, можно получить грубую оценку возможных преимуществ повторением половины рабочего прогона с антитетическим или стратифицированным потоком псевдослучайных чисел. Комбинируя эти наблюдения с наблюдениями, сделанными в первой половине рабочего прогона, мы можем определить, следует ли уменьшать дисперсию и в какой степени. Для оправдания использования методов уменьшения дисперсии в эксперименте всего лишь из 12 прогонов, улучшение должно быть существенным. Если эти результаты успешны для обоих индексов, то нам следует рассматривать в каждом прогоне $h'/2$ антитетических или стратифицированных подпрогонов после первых основных $h'/2$ подпрогонов. Разумеется, требования заданной точности выполняются при значениях h', меньших h. Антитетические или стратифицированные потоки могли бы начинаться с того же начального значения, что и основной.

Более стандартным применением этих методов был бы повторный запуск второй половины каждого прогона с тем же начальным состоянием, что и у первой половины. В этом случае эксперимент потребует 24 прогонов, с обработкой в каждом $n_0^*+h'q/2$ заданий.

Другим путем уменьшения общей продолжительности каждого прогона было бы уменьшение дисперсии подвыборочных средних посредством какого-либо метода уменьшения дисперсии внутри каждой подвыборки. Это повлечет за собой переключение на антитетический или стратифицированный поток для событий, относящихся к последним $q/2$ заданиям каждой подвыборки, и обратно, на первоначальный поток, в начале следующей подвыборки. Однако эта схема более громоздка, чем предыдущая, и скорее подходит для экспериментов, в которых дисперсия оценивается методом независимых копий.

3.5. ИНТЕРПРЕТАЦИЯ РЕЗУЛЬТАТОВ МОДЕЛИРОВАНИЯ

Методы, используемые для интерпретации результатов измерений, которые были описаны в разд. 2.7, можно применить без изменения к результатам экспериментов на модели.

При моделировании часто не требуется производить автономного сжатия данных, поскольку большая часть моделирующих программ осуществляет сжатие данных в момент сбора или в конце прогона, когда производится желаемая для экспериментатора выдача данных, или в обоих случаях. Графические методы представления многих индексов производительности, соответствующих одной и той же системе, на одном листе графика так же полезны в исследованиях на модели, как и в измерениях. Построение эмпирических моделей по данным

моделирования часто весьма полезно в процессе получения информации, относящейся к интересующим нас вопросам.

Не только при измерениях, но также (быть может, даже более) при моделировании не следует забывать, что ошибки возможны и, конечно, часты. В эксперименте на модели наиболее распространенными источниками ошибок являются: неточности, внесенные при формулировке модели системы или рабочей нагрузки, логические ошибки или ошибки программирования в моделирующей программе, недостаточная точность результатов экспериментов из-за ошибок в планировании экспериментов, неправильных статистических гипотез, плохих генераторов псевдослучайных чисел, из-за ошибок в проектировании или написании вспомогательных программ сбора данных в модели, из-за начального и (или) конечного смещений. Легко упустить некоторые, даже существенные, детали при работе со сложными задачами, возникающими при моделировании вычислительной системы. Поэтому оценщик никогда не должен принимать вслепую результаты, выдаваемые моделью, и основывать на них важные решения. В любом случае никогда не помешает осторожность, подозрительное отношение к данным, полученным при прогоне модели или из сеанса измерения. Для повышения уверенности в принимаемых решениях надо по возможности прибегать к помощи экспертов по системам для проверки разумности результатов, применять альтернативные методы получения тех же самых данных, проводить сравнения с результатами предыдущих работ такого же типа, а также использовать методы статистического анализа данных. Это особенно полезно в тех случаях, когда исследование может дать техническую информацию для принятия важного решения.

Задачи

3.1. Нарисуйте карту событий для модели ПРОП-системы, описанной в примере 3.1. Вычертите блок-схему программы управления моделью и всех программ обработки событий, появляющихся на карте событий.

3.2. Предположим, что для обеспечения быстрого прохождения коротких заданий, ПРОП-система, описанная в примере 3.2, назначает каждому заданию квант времени ЦП. Как только время непрерывного обслуживания некоторого задания центральным процессором достигает величины кванта, выполнение этого задания приостанавливается до тех пор, пока не будет исчерпана очередь заданий к ЦП, после чего указанное задание получит новый квант. Приостановленное задание не разгружается из памяти, а просто ставится вновь в очередь к ЦП. Измените карту событий на рис. 3.8 так, чтобы отразить это изменение системы, и нарисуйте блок-схемы программ обработки событий, которые подверглись при этом изменению, если такие окажутся.

3.3. Повторите пошаговое моделирование, описанное в табл. 3.3, предполагая, что первоначально список событий содержит также следующие события:

Тип события	Номер задания	Время события
УЧ_ОСВОБОЖДЕНИЕ	2	1723
ДИСК_ОСВОБОЖДЕНИЕ	8	1739
(шаг задания завершен)		

Определите для каждого шага содержимое списка событий, трех очередей в табл. 3.3 и очереди к памяти. Первоначально задания 3, 5, 7, 8, 9 и 11 находятся в памяти, а задание 4 ожидает в очереди к памяти.

3.4. Если модель, построенная в примере 3.2, основана на подходе «три программы на ресурс», иллюстрируемом рис. 3.7 (a), вместо подхода «две программы на ресурс» (рис. 3.7 (b)), как будет отличаться содержимое таблицы 3.3 от приведенного?

3.5. Постройте подробную блок-схему программы ЦП_ЗАПР/НАЗН для модели системы, в которой используется приоритетное планирование ЦП. Каждое задание имеет назначенный извне приоритет. Если в очередь к ЦП поступает задание, приоритет которого выше, чем у выполняемого в настоящий момент, выполняемое задание приостанавливается и поступает обратно в очередь к ЦП, а задание с более высоким приоритетом начинает обрабатываться.

3.6. Модифицируйте модель, построенную в примере 3.2 так, чтобы учитывалось влияние на производительность ЦП прерываний, генерируемых при завершении каждого обмена с диском или барабаном. Опишите изменения, если они будут необходимы, которые следует сделать в карте событий на рис. 3.8, в программах обработки событий, в формулах в табл. 3.5, в параметрах рабочей нагрузки в табл. 3.6. Если во время построения модели вы поймете, что такая модификация может оказаться желательной, измените ли вы структуру моделирующей программы? Если да, то как?

3.7. Придумайте программы ПОСТУПЛЕНИЕ и УХОД для модели с планированием событий в диалоговой системе и покажите, чем они отличаются от описанных в примере 3.2 для системы с пакетной обработкой. В этой задаче от вас не требуется выбрать и описать специфическое представление рабочей нагрузки системы, но вы должны предположить, что задано описание поведения пользователя, например в терминах распределения или последовательности времен обдумывания.

3.8. Повторите задачу 3.1, предполагая, что модель управляется последовательностью событий и что эта последовательность содержит адреса на барабане, заданные в виде пар (номер дорожки, номер сектора), и количества слов, передаваемые при каждом обращении к барабану, вместо того, чтобы содержать просто времена запросов на барабан.

3.9. Придумайте структуру модели ПРОП-системы, описанной в примере 3.1, основываясь на

a) подходе с последовательным просмотром видов деятельности,

b) подходе со взаимодействием процессов (в любой версии, какую вы предпочтете).

В обоих случаях вы должны нарисовать блок-схему всей модели и детализировать, насколько сумеете, структуры ее программной части и данных.

3.10. Зависит ли модель рабочей нагрузки от определенного подхода (планирование событий, просмотр видов деятельности, взаимодействие процессов), выбранного при проектировании модели системы? Например, полагаете ли вы, что описания рабочей нагрузки для двух моделей одной системы, описанных в задаче 3.9, будут отличаться друг от друга или обе эти модели могут управляться в точности одной моделью рабочей нагрузки? Почему?

3.11. Сконструируйте модель для диалоговой ПРОП-системы, описанной в примере 1.1 (см. рис. 1.6). Нарисуйте карту событий и разработайте модель рабочей нагрузки, выбрав один из трех обсужденных в разделе 3.2.4 типов представления. Задайте также организацию данных системы и рабочей нагрузки, включая очереди и список событий, и выразите в форме блок-схемы действия каждой программы обработки события.

3.12. Параметры рабочей нагрузки, перечисленные в табл. 3.6, не единственный набор параметров, который можно использовать для представления данной модели рабочей нагрузки. Предложите один или два альтернативных набора параметров. Набор следует рассматривать как альтернативный, если он отличается от данного хотя бы одним параметром.

3.13. Придумайте схему измерения средней длины очереди к памяти на протяжении всего прогона модели, разработанной в примере 3.2. Опишите изменения, которые вы бы сделали в модели, чтобы вставить вашу схему и тесты, которые понадобились бы для проверки, корректна ли схема и правильно ли она запрограммирована.

3.14. Спроектируйте представление рабочей нагрузки для модели, такое, чтобы его можно было описать как частично программно-управляемое, а частично управляемое распределением. Оправдайте ваш выбор представления, показав, что в некотором исследовании оно может быть более удобным, чем чистые представления в форме программы или распределения.

3.15. Нарисуйте карту событий для модели системы, изученной в задаче 1.7 и перечислите набор параметров рабочей нагрузки, необходимых для такой модели.

3.16. Аналитическая модель барабана позволяет нам выразить при определенных предположениях для стратегии планирования барабана «наименьшее время доступа — в первую очередь» среднее время общего обслуживания запроса на барабан как [см. Коффман и Деннинг (1973), стр. 209]

$$\bar{t}_{\text{бр}} = \frac{2N + s - 1}{2} \cdot \frac{R_{\text{бр}}}{s},$$

где N — общее число запросов на барабан в системе, $R_{\text{бр}}$ время оборота барабана, а s — число секторов. Заметим, что $\bar{t}_{\text{бр}}$ включает также время ожидания запроса в соответствующей очереди секторов. Кроме того, мы можем рассматривать значение N, предполагаемое постоянным при дифференцировании этой формулы, как число запросов на барабан в момент поступления интересующего нас запроса. Если модель из примера 3.2 должна быть сконструирована с использованием этой модели барабана, какие изменения следует внести в конструкцию? Какие изменения следует внести в табл. 3.6?

3.17. Просмотрите внимательно табл. 3.5 и скажите, какие калибровочные параметры вы ввели бы в формулы запросов ресурсов. Попытайтесь обосновать ваш выбор разумными аргументами.

3.18. Придумайте модель диалоговой системы, в которой каждый пользователь может вводить три класса команд. Каждый класс характеризуется своими распределениями длин входных сообщений, времен ЦП, длин выходных сообщений и времен обдумывания после каждого выходного сообщения. Переходы между классами команд для каждого пользователя управляются заданной матрицей вероятностей переходов размера 4×4. Четвертое состояние пользователя — состояние бездействия (отключения от системы). Распределение продолжительностей состояния бездействия также задано. Все пользователи обладают статистически идентичным поведением. Рассматриваются три класса команд: команды редактирования, компиляции и манипулирования файлами. Конфигурация системы включает ЦП, основную память и диск. Редактор и компилятор постоянно расположены в основной памяти. Нарисуйте карту событий модели и подробные блок-схемы подпрограмм ПОСТУПЛЕНИЕ и ЦП_ЗАПР/НАЗН. Предполагая, что модель должна моделировать существующую установку в целях улучшения производительности, изложите, как вы стали бы ее отлаживать и тестировать и определите критерий калибровки для нее.

3.19. Спланируйте эксперимент для модели, сформулированной в задаче 3.18. Эксперимент должен определить зависимость среднего времени ответа системы от числа терминалов пользователей и от продолжительности кванта времени. Определите план эксперимента и сообщите, как вы выбирали бы продолжительности прогонов модели.

4
Аналитические методы

4.1. АНАЛИТИЧЕСКИЕ ИССЛЕДОВАНИЯ

Согласно определению, данному в разд. 1.6, модельный подход к оценке производительности вычислительной системы называется аналитическим, если модель решается методом, отличным от имитационного моделирования.

Фаза планирования аналитического исследования состоит из следующих трех основных операций:

a) формулировка модели,
b) решение модели,
c) калибровка и контроль модели.

Эти операции аналогичны операциям, проводимым при имитационном моделировании. Сначала должна быть разработана модель системы и ее рабочей нагрузки. Затем в отличие от имитационной модели, которая должна быть приведена в действие, аналитическая модель должна быть решена в численном или аналитическом виде в зависимости от того, какие нужно получить результаты.

Иногда можно воспользоваться уже существующей моделью, аналитическое или графическое решение которой известно. В этом случае, как и при использовании коммерческой моделирующей программы в имитационном исследовании, можно получить ответы простой подстановкой в модель характеристик исследуемой установки в качестве значений входных параметров.

Для того чтобы результаты модельных исследований были достоверны и надежны, точность модели должна быть проверена и, если необходимо, сделана приемлемой с помощью процедур калибровки и контроля.

Точность программной модели была определена в разд. 3.1 и 3.3.1. Это же определение, поясненное на рис. 3.1, может быть применено и к аналитическим моделям. Модель является *точной* при заданном входном условии, если выработанные ею значения индексов производительности достаточно близки к значениям этих индексов для самой системы при том же входном условии.

После того как будут определены максимальные допустимые ошибки индексов производительности, можно говорить, что модель *пригодна* при некотором входном условии, если эти ошибки удовлетворяют требованиям точности. Область пригодности модели есть множество всех входных условий, для которых модель является пригодной.

Калибровка — это проверка пригодности модели для ограниченного числа входных условий, за которой при необходимости следует модификация модели, делающая ее точность приемлемой. Эта проверка должна по возможности производиться сравнением значений выходных переменных модели с замеренными данными. Операция калибровки всегда должна проводиться перед использованием модели для оценки производительности.

Поскольку модель в основном используется при входных условиях, не совпадающих с теми, для которых она калибрована, очень важно убедиться в том, что изменение входных условий не слишком влияет на ее точность. Модель называется *устойчивой*, если ее точность относительно нечувствительна к изменению входных условий, иначе говоря, если ее область пригодности достаточно велика. Устойчивость модели зависит, вообще говоря, от ее внутренней точности и от того как она калибрована. Поэтому устойчивость модели можно определить только с учетом входных условий и требований точности, используемых при ее калибровке. Проверка и при необходимости улучшение устойчивости модели являются задачами процедуры контроля модели.

Как и для имитационной модели, точность аналитической модели зависит от степени детальности, с которой она сформулирована, от способа решения и от точности вводимых значений параметров. Имитационная модель почти всегда дает ошибки, которые порождаются при «решении» модели, например за счет несоответствия оценок индексов производительности, а также начальных или конечных смещений. С другой стороны, для некоторых, самых простых с математической точки зрения, аналитических моделей существует точное решение. Это замечание не должно заставить нас рассматривать такие модели, как более надежные, чем те, которые разрешимы только с помощью приближенных методов. На самом деле их математическая простота обычно является следствием сравнительно низкой степени детальности, что приводит к относительной неточности этих моделей.

Очень важно различие между детерминированными и вероятностными моделями. *Детерминированные модели* оказываются полезными во многих областях машинной математики и вычислительной техники. К этой категории относятся машины Тьюринга, разного рода автоматы, графические модели программ, сети Петри. Несмотря на то что некоторые из этих моделей могут оказать существенную помощь в понимании вычислительных систем и их компонентов и в создании их имитационных моделей, пока их приложения в качестве аналитических моделей для оценки производительности систем ограничены.

Одним из главных препятствий для использования детерминированных моделей в исследованиях оценки производительности является их относительная неспособность отображать изменчивость рабочей нагрузки, наблюдаемую в большинстве вычислительных установок. Описание поведения задания несколькими детерминированными параметрами требует того, чтобы были сделаны очень грубые приближения. Поэтому рабочая нагрузка может быть смоделирована с помощью

набора детерминированных параметров, описывающих ее задания, только для тех исследований или для тех установок, для которых такие приближения приемлемы. Часто приходится делать дополнительное предположение, что все задания поступают в одно и то же время или с постоянной частотой.

Детерминированная модель рабочей нагрузки применима также в том случае, если делается предположение, что рабочая нагрузка состоит из нескольких заданий, которые возникают периодически. Тогда поведение этих заданий может быть охарактеризовано с большей детальностью, чем если бы рабочая нагрузка была непериодической. Частным случаем, для которого детерминированная модель обычно легко применима, является периодическая рабочая нагрузка, состоящая из одинаковых заданий. Такая модель рабочей нагрузки не слишком далека от действительности для некоторых специализированных установок.

Подводя итоги, можно сказать, что детерминированные модели, по крайней мере в настоящее время, могут быть использованы для оценки порядка величины или границ некоторых индексов производительности, для исследования некоторых задач распределения ресурсов или для разработки и оценки некоторых систем, рабочую нагрузку которых удается сравнительно удобно характеризовать детерминированным образом.

Отличие *вероятностных моделей* состоит в том, что они естественным образом подходят для представления некоторых аспектов изменчивости рабочей нагрузки. К сожалению, другие аспекты, такие, как порядок появления заданий и корреляция между случайными величинами, характеризующими рабочую нагрузку, или невозможно представить в контексте вероятностной модели, или их представление делает решение модели очень сложным. Поэтому эти аспекты обычно игнорируются.

Вероятностные модели, основывающиеся на теории очередей, особенно удобны для исследования перегруженных вычислительных систем, в которых за обладание ресурсами соревнуются несколько процессов. Существующие математические методы позволяют исследователю решить модель, только когда она достаточно проста. Для того чтобы модель стала разрешимой, обычно делаются некоторые упрощающие предположения, и эти предположения могут оказаться причиной того, что точность модели станет неудовлетворительно низкой.

В общем случае очень мало известно о чувствительности индексов производительности относительно некоторых упрощений, принимаемых в аналитических моделях. Немногие исследования, в которых результаты, полученные аналитическими методами, были сопоставлены с измеренными значениями индексов производительности, продемонстрировали удивительное согласие действительной производительности системы с предсказанной довольно простыми и грубыми моделями. Два примера подобных исследований обсуждаются в разд. 4.4 [см. Шерр (1967)], 7.4.1 и 7.4.3 [см. Хьюз и Мо (1973)].

Аналитическая модель разрешима в *замкнутой форме* относительно индекса производительности, если ее решение позволяет выразить этот индекс символически как явную функцию от параметров модели. Существование решения в замкнутой форме для некоторого индекса не означает, что другие индексы также могут быть выражены в замкнутой форме.

Читатель должен отметить, что модель, не разрешимая в замкнутой форме относительно некоторого индекса, может все же оказаться разрешимой относительно этого индекса. Класс моделей, для которых решение известно в замкнутой форме, существенно уже класса разрешимых моделей. И хотя в большинстве случаев главное значение имеет сам факт разрешимости, тем не менее решение в замкнутой форме более предпочтительно, чем численное, поскольку оно позволяет вычислить значения нужного индекса производительности быстрее и экономней. Кроме того, такие решения позволяют лучше понять зависимости между индексами производительности системы и параметрами рабочей нагрузки.

Некоторые модели допускают решение в форме, которую можно назвать *частично замкнутой*. Это имеет место в том случае, когда сама математическая структура формулы, выражающей интересующий нас индекс, зависит от одного или более параметров модели. Может оказаться, например, что имеется некоторое количество аналитических выражений в замкнутой форме для данного индекса, и мы всякий раз должны выбирать одно из них в зависимости от значений определенных параметров. Два примера решений в частично замкнутой форме даны в разд. 4.3.5.

Полезно помнить еще и о различии между *точным* и *приближенным* решениями. Когда не может быть найдено точное решение, то вполне приемлемо и приближенное решение, если вносимые им в решение погрешности находятся в допустимых пределах. Приближенное решение в замкнутой форме иногда может оказаться предпочтительней точного решения, которое требует использования численного приближенного метода, поскольку не может быть выражено в замкнутой форме. Для вероятностных моделей пока разработано только несколько приближенных методов, но в этой области ожидается быстрый прогресс. Два основных класса методов, приложимых к моделям с очередями, это *диффузионная аппроксимация* и *декомпозиция*. Основные концепции, на которых основаны эти методы, коротко резюмированы в конце разд. 4.3.4.

Использование аналитических моделей в исследованиях по оценке производительности до сих пор сильно ограничивалось трудностями решения близких к реальности моделей и недостатком доверия со стороны большинства специалистов к более простым, разрешимым моделям. Этот недостаток доверия не всегда оправдан. Аналитические методы гораздо чаще применяются для исследования компонентов системы и для выработки стратегии распределения услуг, чем для оценки вычислительных систем в целом. Эта ситуация, которая частично от-

ражает правильную оценку состояния дел, а частично обусловлена трудностями психологического характера, должна заметно измениться в ближайшее время, как только улучшатся состояние дел и понимание аналитических методов.

4.2. ДЕТЕРМИНИРОВАННЫЕ МОДЕЛИ

4.2.1. Введение

Вычислительные системы и их рабочие нагрузки очень часто оказываются слишком сложными, чтобы их можно было достаточно точно представить с помощью обозримой детерминированной модели. Рабочая нагрузка вычислительной установки может быть охарактеризована ограниченным числом детерминированных параметров только в таких исследованиях, для которых приемлемо очень грубое приближение, или для исследования частных видов установок. Некоторые устройства, используемые в режиме реального времени, например для выборки информации или управления процессами, имеют рабочие нагрузки, детерминированное представление которых может быть сделано как обозримым, так и достаточно точным.

Даже в таких приложениях, в которых рабочую нагрузку составляют задания с одинаковыми или очень сходными характеристиками запросов на ресурсы, структура появления заданий часто более точно моделируется стохастическим процессом, чем простым детерминированным, таким, например, как периодический.

Детерминированные подходы применялись для определения и доказательства свойств некоторых подсистем или алгоритмов управления ресурсами. Такие модели не будут рассматриваться в данной книге. Отметим два важных приложения детерминированных моделей в этих областях: оптимальное планирование работы мультипроцессорных систем [см., например, Конвей и др. (1967) и гл. 3 в книге Коффмана и Деннинга (1973)] и анализ конкретных алгоритмов планирования для внешних запоминающих устройств [см., например, Фуллер (1972)].

Детерминированные модели находят также применение при выводе формул первого приближения и при оценке порядка величины простых индексов производительности вычислительных систем. Эта категория моделей включает *модели средних значений* систем, которые можно считать полученными из вероятностных моделей с помощью замены каждого случайного параметра его средним значением.

Модель средних значений рабочей нагрузки состоит из множества одинаковых заданий, причем характеризующие их параметры совпадают со средними значениями параметров заданий моделируемой рабочей нагрузки. Эти одинаковые задания появляются в модели с постоянной интенсивностью, равной средней интенсивности появления для реальной рабочей нагрузки. Иначе говоря, задания не появляются и не исчезают, а постоянное число одних и тех же заданий непрерывно циркулирует в системе. В этом случае, особенно когда число заданий невелико, допущение об идентичности заданий можно ослабить.

На моделях средних значений обычно получаются оптимистические результаты. Другими словами, как мы покажем в разд. 4.4, предсказанные значения индексов производительности в основном лучше действительных.

Более точные оценки иногда могут быть получены на моделях, являющихся частично вероятностными, а частично моделями средних значений. Примером могут служить модели с очередями для систем с виртуальной памятью, в которых страничный режим подкачек представляется аналитическим приближением кривой, выражающей среднюю интенсивность страничных отказов в зависимости от отведенного программе объема памяти (см. пример 7.3). Это также относится к детерминированным моделям, использующим для представления отдельных частей системы результаты, полученные на моделях с очередями. Модель такого типа описана у Чемберлена и др. (1973).

Детерминистские подходы к моделированию вычислительных систем могут быть использованы для исследования наилучших и наихудших случаев, в которых каждый случайный параметр заменяется одним из своих крайних значений. Крайнее значение можно определить как среднее значение плюс или минус среднее квадратичное отклонение. Полученные модели можно назвать *моделями крайних значений*.

В разд. 4.2.2 и 4.2.3 на двух простых примерах детерминированных моделей мы поясним приближения, которые должны быть введены для формулировки этих моделей, приемы, которые должны применяться при написании уравнений, и результаты, которые можно получить.

4.2.2. Модель перекрытия работы центрального процессора и ввода-вывода

В этом разделе мы представим детерминированную модель однопрограммной системы, предложенную Хеллерманом и Смитом (1970). Модель предназначена для оценки и сравнения различных схем перекрытия работы ЦП и ввода-вывода. Так как моделируется однопрограммная система, то рассматривается перекрытие между работой центрального процессора и ввода-вывода для одного и того же задания.

Пример 4.1. Система, которую мы хотим представить, состоит из центрального процессора, основной памяти, одного или нескольких устройств внешней памяти и одного или нескольких независимых каналов между устройствами внешней памяти и основной памятью. Предполагается, что рабочая нагрузка состоит из единственного задания, которое бесконечно обрабатывает записи из базы данных, расположенной во внешней памяти. Записи для обработки должны быть загружены в основную память. После окончания обработки записи снова запоминаются во внешней памяти. Пересылка информации между основной и внешней памятью осуществляется не записями, а блоками. Все записи имеют один и тот же размер, обозначенный r. Блоки тоже имеют постоянный размер b, кратный r. Коэффициент *блокирования* определяется как

$$g = \frac{b}{r}. \tag{4.1}$$

Основная память делится на v областей, каждая из которых содержит блок. Таким образом,

$$v=\frac{M}{b}, \tag{4.2}$$

где M — величина той части основной памяти, в которую могут загружаться записи.

В то время, как центральный процессор обрабатывает блок, расположенный в одной из областей основной памяти, остальные области используются каналами ввода-вывода как буферные для операций ввода-вывода. Внешняя память и каналы ввода-вывода характеризуются постоянным временем доступа t_a к каждому блоку и временем пересылки слова t_w. Таким образом, время пересылки записи и блока равны соответственно rt_w и bt_w.

Время, необходимое центральному процессору для обработки записи, предполагается одинаковым для всех записей и обозначается как $t_{цп}$. Индекс производительности, вычисляемый при использовании данной модели,— это пропускная способность T, выраженная числом записей, обработанных за единицу времени. Если единица времени много больше чем $t_{цп}$, то пропускная способность постоянна. Введенные нами модельные параметры, а также их числовые значения, которые мы будем использовать при вычислениях, приведены в табл. 4.1. Так как b и r предполагаются фиксированными, то значение M должно меняться при изменениях v, как показано в (4.2) и табл. 4.1.

Таблица 4.1

Параметры детерминированной модели перекрытия центрального процессора и ввода-вывода, рассматриваемой в примере 4.1

Название параметра	Обозначение	Значение
Размер основной памяти (область данных)	M	$2K$, $4K$, $6K$ слов
Размер записи	r	64 слова
Размер блока	b	$2K$ слова
Время обработки одной записи процессором	$t_{цп}$	1.6 мс
Время пересылки слова из внешней памяти	t_w	10 мкс
Время доступа к внешней памяти	t_a	15 мс
Число областей (блоков) в основной памяти	v	1, 2, 3

В табл. 4.2 приведены четыре схемы перекрытия, которые мы хотим оценить. *Степень* перекрытия совпадает с числом областей в основной памяти.

В *последовательной* схеме, изображенной на рис. 4.1 (a), перекрытие отсутствует. Блок вводится (ось *I*), проходит фазу вычислений (ось *C*) и выводится (ось *O*). Поскольку в памяти имеется только одна область ($v=1$), следующий блок не может быть загружен до тех пор, пока предыдущий не будет полностью выведен во вспомогательную память.

Времена ввода, вычисления и вывода блока могут быть записаны следующим образом:

$$\begin{aligned} t_I &= t_a + bt_w, \\ t_C &= gt_{цп}, \\ t_O &= t_a + bt_w = t_I. \end{aligned} \tag{4.3}$$

Простое изучение временной диаграммы на рис. 4.1 (a) позволяет выразить время обработки одной записи $1/T$ как функцию от параметров модели. Определим *время обработки блока* t_B как отношение длины сравнительно большого периода времени к числу блоков, обработанных системой в течение этого периода. Ясно, что $1/t_B$ —

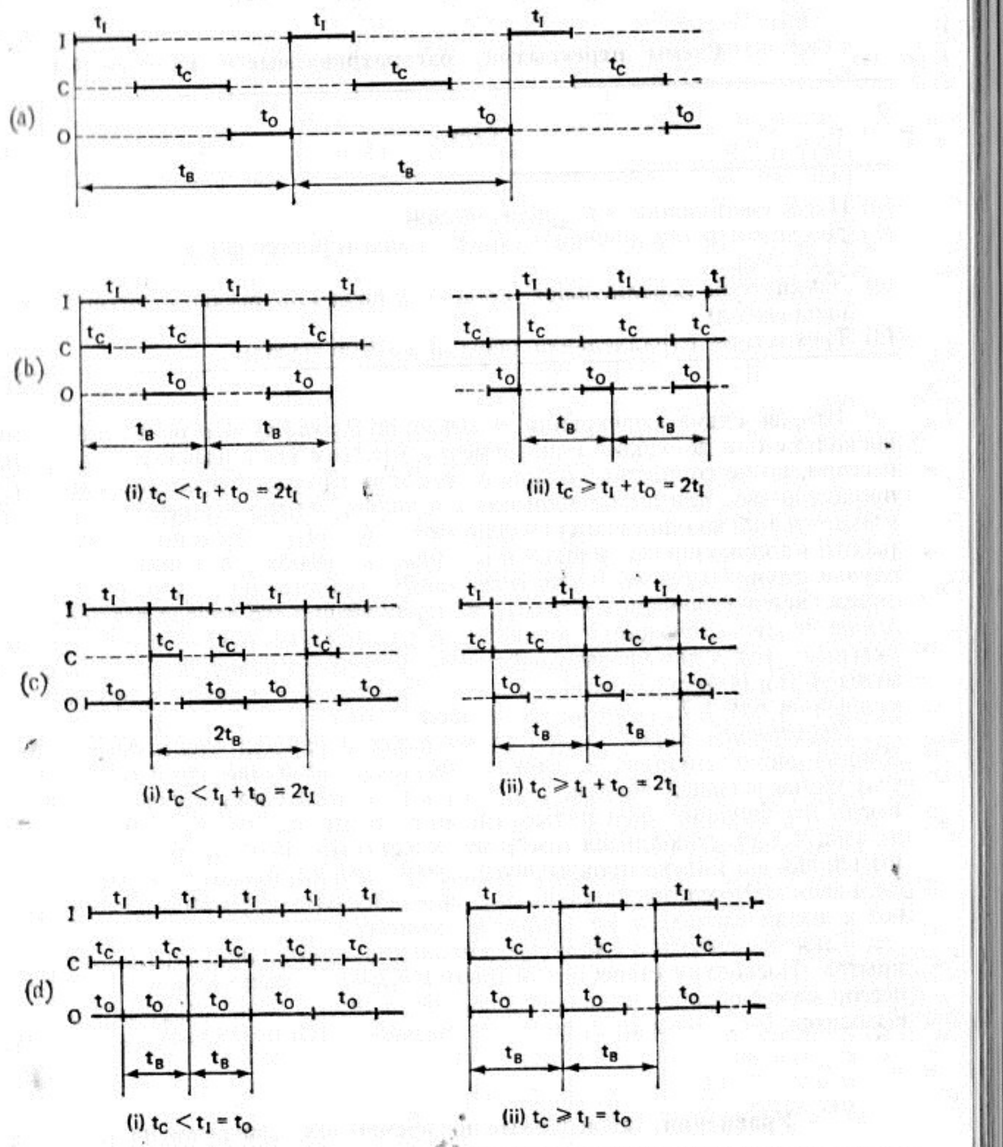


Рис. 4.1. Временные диаграммы для четырех схем перекрытия: (a) — последовательная схема, (b) — двухпутная ограниченно параллельная схема, (c) — двухпутная параллельная схема, (d) — трехпутная параллельная схема. Для последних трех схем (i) — случай ограничения по вводу-выводу, (ii) — случай ограничения по вычислениям.

это пропускная способность системы, выраженная в блоках за единицу времени. Таким образом,

$$\frac{1}{T}=\frac{t_B}{g}, \tag{4.4}$$

поскольку в каждом блоке содержится g записей. По рис. 4.1 (a) t_B равно $t_I+t_C+t_O$. Согласно (4.3), эту сумму можно записать как $gt_{цп}+2t_a+2bt_w$. Это выражение, деленное на g как предложено в (4.4), дает уравнение для $1/T$ (см. табл. 4.3 (a)).

Таблица 4.2

Схемы перекрытия, рассматриваемые в примере 4.1

Схема	Степень перекрытия
(a) Последовательная (нет перекрытия)	1
(b) Двухпутная ограниченно параллельная (процессор-ввод или вывод-процессор)	2
(c) Двухпутная параллельная (процессор-ввод или вывод-процессор или ввод-вывод)	2
(d) Трехпутная параллельная (ввод-процессор-вывод)	3

Вторая схема перекрытия называется *двухпутной ограниченно параллельной*, поскольку она допускает одновременную работу ввода-вывода и центрального процессора, но не совместные операции ввода и вывода. Разрешаются перекрытия типа процессор-ввод или вывод-процессор, но не ввод-вывод. Степень перекрытия равна 2, и имеется только один канал ввода-вывода, который во всякий момент времени может работать только в режиме ввода или в режиме вывода. Должны быть рассмотрены два случая: случай *ограничения по вводу-выводу*, изображенный на рис. 4.1 (b) (i), который определяется условием $t_C < t_I + t_O$, и дополняющий его случай *ограничения по вычислению*, соответствующий условию $t_C \geqslant t_I + t_O$ (см. рис. 4.1 (b) (ii)). Из этих временных диаграмм, как и для предыдущей схемы, получим выражения для t_B. Поскольку формулы (4.3) и (4.4) справедливы и для этой схемы (как и для всех других), мы получим уравнения для $1/T$, приведенные в табл. 4.3 (b).

Двухпутная параллельная схема характеризуется степенью перекрытия $v=2$. Данная система, однако, содержит два канала ввода-вывода и по крайней мере два устройства внешней памяти, так что ввод и вывод могут осуществляться совместно. Когда это случается, центральный процессор остается незагруженным. Как показано на рис. 4.1 (c), дополнительные возможности этой системы по сравнению со схемой (b) влияют на индекс производительности системы только в случае ограничения на ввод-вывод. Это следовало ожидать, так как в схеме (c) увеличивается производительность ввода-вывода, а не скорость процессора.

Такие же выводы легко получаются и для *трехпутной* параллельной схемы перекрытия. Поскольку степень перекрытия в данной схеме равна 3, центральный процессор может обрабатывать один блок, пока другой блок загружается, а третий блок выводится. Эта схема требует тех же возможностей ввода-вывода, что и схема (c), но

Таблица 4.3

Уравнения, выведенные из временных диаграмм на рис. 4.1

Схема	Случай	Уравнения	
(a)	—	$1/T = t_{цп} + 2t_a/g + 2rt_w$	$z = 1 + 2y + 2x$
(b)	(i)	$1/T = 2t_a/g + 2rt_w$	$z = 4y + 2x$
	(ii)	$1/T = t_{цп}$	$z = 1$
(c)	(i)	$1/T = t_{цп}/2 + t_a/g + rt_w$	$z = 1/2 + 2y + x$
	(ii)	$1/T = t_{цп}$	$z = 1$
(d)	(i)	$1/T = t_a/g + rt_w$	$z = 3y + x$
	(ii)	$1/T = t_{цп}$	$z = 1$

...большого объема основной памяти, если размер блока такой же, как в других схемах. Когда центральный процессор насыщен (случай ограничения по вычислению), производительность системы не отличается от производительности, даваемой схемой (c) или (b). В случае ограничения по вводу-выводу, как показано в табл. 4.3 (d) (i), время обработки записи на $t_{цп}/2$ единиц времени меньше, чем соответствующее время в схеме (c).

В табл. 4.4 приведены значения пропускной способности T и коэффициента использования ρ_1 центрального процессора P_1, даваемые четырьмя рассмотренными выше схемами, при значениях параметров модели, приведенных в табл. 4.1. Значения

Таблица 4.4

Результаты, полученные из модели, описанной в примере 4.1, и ее параметров из табл. 4.1

Схема	Постоянное b, переменное M			Постоянное M ($2K$ слов), переменное b		
	Случай	T (запись/с)	ρ_1	Случай	T (запись/с)	ρ_1
(a)	—	261.9	0.419	—	261.9	0.419
(b)	(i)	450.9	0.721	(i)	317.2	0.507
(c)	(i)	523.9	0.838	(i)	420.6	0.672
(d)	(ii)	625	1	(i)	488.7	0.781

ρ_1 получаются как частные от деления T на $T_{max}=625$ записей в секунду. Поскольку модель имеет решения в аналитической форме, которые представлены уравнениями, приведенными в табл. 4.3, вычисление значения индекса производительности, соответствующего любой заданной комбинации параметров модели, является очевидным.

Также просто исследовать поведение индекса производительности как функции от параметров. Мы можем, например, задаться вопросом, какая из схем перекрытия является лучшей в случае, когда объем памяти M и размер записи r фиксированы, а размер блока b изменяется вместе с v согласно (4.2). Чтобы ответить на этот вопрос, нормализуем время обработки записи, разделив его на $t_{цп}$, т. е. на время, затрачиваемое на обработку одной записи центральным процессором. Это отношение

$$z = \frac{1}{T t_{цп}} \tag{4.5}$$

равно единице, только если перекрытие между работой центрального процессора и вводом-выводом абсолютно, а коэффициент использования центрального процессора равен 1. Чаще, особенно в случае ограничения по вводу-выводу, z больше 1. На самом деле можно легко показать, что z является величиной, обратной к ρ_1, коэффициенту использования процессора.

Запишем теперь нормализованные уравнения из табл. 4.3 как функции от следующих двух параметров:

$$x = \frac{rt_w}{t_{цп}}, \quad y = \frac{rt_a}{Mt_{цп}}. \tag{4.6}$$

Заметим, что эти параметры не зависят от величин b или v и что в силу (4.2) и (4.6) $t_a/rt_{цп} = vy$. Новые уравнения также приведены в табл. 4.3. Диаграмма на рис. 4.2 содержит ответ на поставленный выше вопрос. Величины x и y, соответствующие величинам рассматриваемой задачи, изображаются точками в первом квадранте плоскости (x, y). На диаграмме дана классификация по производительности четырех схем перекрытия в зависимости от значений x и y. Например, значения параметров модели в табл. 4.1 (где $M=2K$ слов) дают $x=0.4$ и $y=0.293$. Эта точка попадает в ту

область диаграммы рис. 4.4, которая охарактеризована как (d, c, b, a), в порядке убывания производительности. Это подтверждается значениями T и ρ_1 из табл. 4.4, из которой видно, что схема (d) является лучшей для нашей системы. Фиксация размера памяти в $2K$ слов вызывает убывание T и ρ_1 для всех схем, кроме схемы (a), для

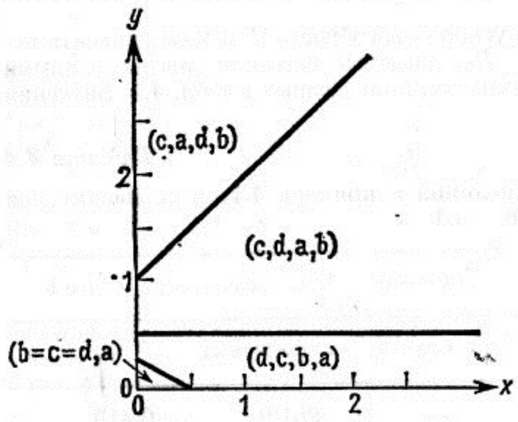


Рис. 4.2. Классификация производительности схем перекрытия для фиксированной памяти и фиксированного размера записи. [*Источник:* Хеллерман и Смит (1970), с разрешения ACM, Inc.]

которой размер памяти не отличается от ранее рассмотренного. Кроме того, уменьшение M с $6K$ слов до $2K$ слов для схемы (d) переводит систему из ограниченной по вычислениям в ограниченную по вводу-выводу, поскольку с уменьшением размера блока t_I и t_O убывают медленнее, чем t_C. Это обусловлено тем, что t_a не зависит от b.

Из рис. 4.2 видно, что усложнение схем перекрытия не всегда приводит к улучшению производительности. Например, в случае когда $y>x+1$, наилучшей является схема (c), а последовательная схема лучше, чем трехпутная параллельная схема или двухпутная ограниченно параллельная схема.

4.2.3. Модель средних значений для мультипрограммной системы

В этом разделе мы коротко опишем простую модель средних значений, предложенную Феничелом и Гроссманом (1969) для оценки пропускной способности мультипрограммной системы. Предполагается, что вычисления и ввод-вывод в процессе никогда не перекрываются. Представленную здесь модель можно рассматривать как частный случай вероятностной модели, обсуждаемой в разд. 4.3.5.

Пример 4.2. Моделируемая система представляет собой простую ПРОП-систему (рис. 1.2), имеющую только одно устройство внешней памяти, скажем барабан. Система является мультипрограммной с постоянной степенью мультипрограммирования N. В любой момент времени всякое задание *активно* или *пассивно*. Задание пассивно или находится в состоянии S_0, если оно не находится в основной памяти. Активное задание находится либо в состоянии S_1, ожидая обработки или обрабатываясь центральным процессором, либо в состоянии S_2, ожидая обработки или обрабатываясь устройством внешней памяти (барабаном). Диаграмма состояний задания изображена на рис. 4.3(a). Предполагается, что как только некое задание становится пассивным, другое задание немедленно активизируется, с тем чтобы степень мультипрограммирования была постоянной.

Все задания предполагаются одинаковыми. Каждое характеризуется требуемым для него временем центрального процессора $t_{цп}$ и числом запросов барабана $n_{бр}$. Эти запросы выдаются через регулярные интервалы процессорного времени, т. е.

ный раз, когда задание обрабатывается процессором в течение $t_{цп}/(n_{бр}+1)$ единиц времени, начиная с окончания последней операции с барабаном или с начала выполнения задания. Каждый запрос барабана приводит к пересылке блока, размер которого фиксирован и одинаков для всех блоков. В конце $(n_{бр}+1)$-го процессорного интервала задание прекращается и становится пассивным. Диаграмма состояний задания на рис. 4.3(a) естественным образом приводит к модели системы, изображенной на рис. 4.3(b). Обозначим через $\bar{N}_1$ среднее число заданий в состоянии S_1 и $\bar{N}_2$ — среднее число заданий в состоянии S_2. Тогда имеем

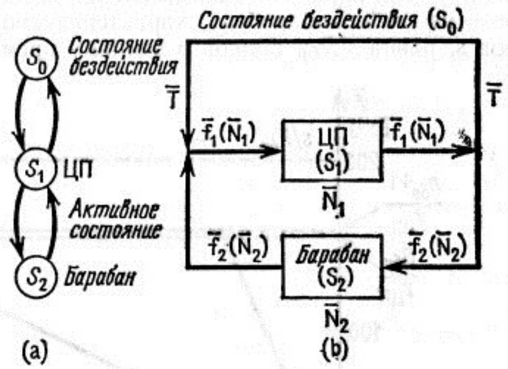


Рис. 4.3. Модель средних значений мультипрограммной системы: (a) — диаграмма состояний для одного задания, (b) — модель системы.

$$\bar{N}_1+\bar{N}_2=N. \tag{4.7}$$

При равновесии средняя интенсивность входного потока должна быть равна средней интенсивности выходного потока как для центрального процессора, так и для барабана. Поскольку ожидается, что эти интенсивности зависят от соответствующих нагрузок $\bar{N}_1$ и $\bar{N}_2$, обозначим их $\bar{f}(\bar{N}_1)$ и $\bar{f}(\bar{N}_2)$. Равенство средних интенсивностей входного и выходного потоков будет выполняться и для пассивного состояния, поскольку N предполагается постоянным. Интенсивности для пассивного состояния на самом деле совпадают с нашим индексом производительности $\bar{T}$ — средней пропускной способностью системы. Каждое задание приходит в состояние S_1 $(n_{бр}+1)$ раз. В первые $n_{бр}$ раз, выходя из состояния S_1, задание попадает в состояние S_2. В последний раз оно попадает в состояние S_0. Таким образом,

$$\bar{f}_2(\bar{N}_2)=\frac{n_{бр}}{n_{бр}+1}\bar{f}_1(\bar{N}_1) \tag{4.8}$$

$$\bar{T}=\bar{f}_1(\bar{N}_1)-\bar{f}_2(\bar{N}_2)=\frac{\bar{f}_1(\bar{N}_1)}{n_{бр}+1}. \tag{4.9}$$

Если $\bar{f}_1(\bar{N}_1)$ известно, то (4.9) позволяет нам вычислить среднюю пропускную способность. Имеем (см. рис. 4.4)

$$\begin{aligned}\bar{f}_1(\bar{N}_1)&=0 && \text{для} \quad \bar{N}_1=0,\\ &=\frac{n_{бр}+1}{t_{цп}} && \text{для} \quad \bar{N}_1\geqslant 1,\\ &=\bar{N}_1\frac{n_{бр}+1}{t_{цп}} && \text{для} \quad 0<\bar{N}_1<1.\end{aligned} \tag{4.10}$$

Первые два случая из (4.10) тривиальны. Когда $\bar{N}_1=0$, все активные задания ожидают барабан, которому требуется бесконечное количество времени для обработки запроса. Таким образом, барабан насыщен, а средняя пропускная способность системы равна 0. Когда $\bar{N}_1\geqslant 1$, процессор всегда занят, и система выдает свою максимальную пропускную способность:

$$\bar{T}_{max}=\frac{1}{t_{цп}}. \tag{4.11}$$

Наиболее интересен последний случай в (4.10), поскольку он соответствует системе, в которой коэффициент использования процессора оценивается числом между 0 и 1. Для решения модели в этом случае нам должна быть известна величина $\bar{f}_2(N_2)$. Эту величину можно определить измерением или моделированием, или из аналитической модели барабана. Максимальная интенсивность выходного потока для разделенного на секторы барабана, характеризуемого временем оборота $R_{бр}$ и числом секторов s, равна $s/R_{бр}$ блоков в единицу времени. Это соответствует пересылке одного

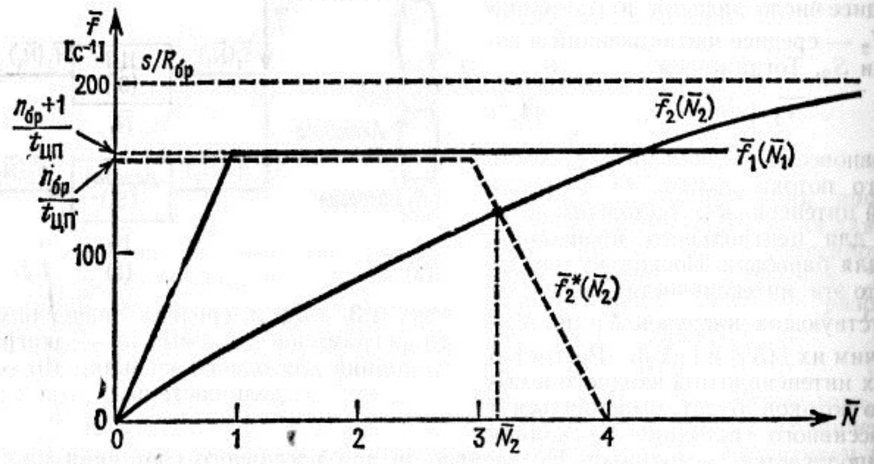


Рис. 4.4. Средние интенсивности входного и выходного потоков для модели на рис. 4.3. Значения параметров те же, что и в табл. 4.5. Пунктирная линия используется в процедуре графического решения модели.

блока в течении периода каждого сектора. Если предположить, что запросы барабана, выдаваемые заданиями, всегда содержат в точности один блок, то имеем

$$\bar{f}_{2\,max} = \bar{f}_2(\infty) = \frac{s}{R_{бр}}. \tag{4.12}$$

Кривая $\bar{f}_2(\bar{N}_2)$ обычно будет выглядеть, как на рис. 4.4. Аналитически постоянный вариант этой кривой можно найти на рис. 4 в книге Коффмана (1969). Введение кривой такого вида, как на рис. 4.4, делает модель нелинейной [$\bar{f}_1(\bar{N}_1)$ кусочно-линейна]. Простой графический метод решения этой модели заключается в построении на плоскости $(\bar{N}_2, \bar{f}_2)$ графика функции

$$\begin{aligned} \bar{f}_2^* &= 0 && \text{для} \quad \bar{N}_2 = N, \\ &= \frac{n_{бр}}{t_{цп}} && \text{для} \quad \bar{N}_2 \leqslant N-1, \\ &= \frac{n_{бр}}{t_{цп}}(N-\bar{N}_2) && \text{для} \quad N-1 < \bar{N}_2 < N. \end{aligned} \tag{4.13}$$

Функция $\bar{f}_2^*$ получается подстановкой (4.7) в (4.10) и (4.10) в (4.8). Точка пересечения $\bar{f}_2^*$ с кривой $\bar{f}_2$ дает значение $\bar{N}_2$, соответствующее решению модели. Отсюда мы можем вывести $\bar{N}_1$, а из (4.9) — $\bar{T}$.

На рис. 4.4 иллюстрируется применение такой процедуры для значений параметров модели из табл. 4.5. Пунктирная линия представляет функцию $\bar{f}_2^*(\bar{N}_2)$. Так как значение $\bar{N}_2$ в точке пересечения равно 3.2, получим $\bar{f}_2(\bar{N}_2)=128$ блоков в секунду, $\bar{N}_1=0.8$, $\bar{f}_1(\bar{N}_1)=131.2$ интервала процессора в секунду и $\bar{T}=3.2$ задания в се-

кунду. Максимальная пропускная способность, задаваемая формулой (4.11), составляет $\bar{T}_{max}=4$ задания в секунду.

Графическое построение на рис. 4.4 дает непосредственное представление о результатах изменения модельных параметров. Например, увеличение N будет увеличивать $\bar{N}_1$ до 1 и $\bar{T}$ до $\bar{T}_{max}$. Тот же результат получается для более быстрого барабана (с меньшим $R_{бр}$) или для более распараллеленного, или для барабана с большей

Таблица 4.5

Параметры модели средних значений мультипрограммной системы, описанной в примере 4.2

Название параметра	Обозначение	Значение
Степень мультипрограммирования	N	4
Процессорное время, запрашиваемое заданием	$t_{цп}$	250 мс
Число запросов барабана заданием	$n_{бр}$	40
Время оборота барабана	$R_{бр}$	20 мс
Число секторов на барабане	s	4

плотностью записей (с большим s), или с лучшим алгоритмом планирования операций с барабаном, который приводит к более быстрому росту кривой $\bar{f}_2(\bar{N}_2)$. Увеличение $\bar{T}$ может также быть получено уменьшением $n_{бр}$. Увеличение $t_{цп}$ будет увеличивать $\bar{N}_1$. Это будет вызывать приближение $\bar{T}$ к $\bar{T}_{max}$, но одновременно уменьшать $\bar{T}_{max}$. Увеличение $t_{цп}$ дальше точки, в которой $\bar{T}$ достигает $\bar{T}_{max}$, дает обратно пропорциональное уменьшение $\bar{T}$. ■

4.3. ВЕРОЯТНОСТНЫЕ МОДЕЛИ

4.3.1. Марковские модели и модели с очередями

Вероятностные модели, благодаря их возможностям представления сложных изменчивых явлений относительно компактным образом, применяются в исследованиях вычислительных систем значительно шире, чем детерминированные модели. Среди вероятностных подходов в аналитическом моделировании, несомненно, более привлекательным и распространенным является метод очередей. Однако марковские или полумарковские модели тоже иногда используются для оценок. Другим, более важным соображением для рассмотрения здесь марковских моделей является то, что их теория служит основой элементарной теории очередей. В этом разделе мы дадим простое описание марковских моделей, перед тем как ввести модели с очередями, составляющие предмет рассмотрения разд. 4.3.2 — 4.3.5.

Марковская цепь, т. е. марковский процесс с дискретными состояниями,— это стохастический процесс $X(t)$ (см. разд. 2.4.3) с состояниями S_0, S_1, S_2, . . . , причем вероятность того, что в момент времени t_{k+1} процесс находится в состоянии S_j, зависит только от его состояния в момент t_k для любой последовательности моментов времени t_1, t_2, . . . , t_{k+1}, такой, что $t_1 < t_2 < \ldots < t_{k+1}$.

Мы говорим, что процесс находится в состоянии S_{h_i} в момент t_i, если $X(t_i)=h_i$, где h_i — неотрицательное целое. Предыдущее определение может быть, таким образом, записано в виде

$$\text{Prob}\,[X(t_{k+1})=j \mid X(t_1)=h_1,\ X(t_2)=h_2,\ \ldots,\ X(t_k)=h_k] = \text{Prob}\,[X(t_{k+1})=j \mid X(t_k)=h_k]. \quad (4.14)$$

Зависимость (4.14) называется *марковским свойством с непрерывным временем* и определяет *марковскую цепь с непрерывным временем*. Выражение *непрерывное время* указывает на то, что переход из состояния в состояние разрешен в любой момент времени. Если ограничить переходы только дискретными моментами времени, обозначенными индексами времени 1, 2, . . . , k, . . . , то можно определить *марковское свойство с дискретным временем* для стохастического процесса X_k как

$$\text{Prob}\,(X_{k+1}=j \mid X_1=h_1,\ X_2=h_2,\ \ldots,\ X_k=h_k) = \text{Prob}\,(X_{k+1}=j \mid X_k=h_k). \quad (4.15)$$

Случайная последовательность X_k, удовлетворяющая (4.15) для всех целых положительных k и всех возможных состояний, называется *марковской цепью с дискретным временем*. Для такой цепи вероятность перехода из состояния S_i в состояние S_j в момент k может быть записана как

$$p_{ij}(k)=\text{Prob}\,(X_{k+1}=j \mid X_k=i). \quad (4.16)$$

Марковское свойство дает возможность определить отношения между состояниями в виде матрицы $\mathbf{P}(k)$ — матрицы вероятностей переходов. Если вероятности переходов не зависят от времени, то можно обозначить их просто p_{ij} и назвать цепь *однородной*. *Диаграмма переходов состояний* марковской цепи — это направленный граф, вершины которого соответствуют состояниям, а ребра — переходам состояний. Каждое ребро помечено значением вероятности соответствующего перехода. Для однородной цепи эти метки не зависят от времени. Цепь полностью определяется ее диаграммой переходов состояний и ее начальным состоянием. Эта диаграмма может рассматриваться как графический эквивалент матрицы вероятностей переходов $\mathbf{P}$.

Моделируя систему с помощью марковской цепи, мы хотим использовать нашу модель, чтобы ответить на некоторые вопросы относительно производительности системы. Нам, например, интересно определить, как часто система оказывается в определенном состоянии, сколько времени система будет находиться в этом состоянии, каков средний интервал времени между приходами системы в одно состояние. Если марковская модель удовлетворяет определенным условиям, которые будут определены ниже, аналитический ответ на эти вопросы относительно прост. Марковская цепь *неприводима*, если любое состояние может быть достигнуто из любого другого состояния (на диаграмме переходов состояний должен существовать направленный путь, ведущий из состояния S_i в состояние S_j для всех i и j). Состояние *возвратно*, если вероятность вернуться в него после посещения равна 1. *Среднее время возвращения* $\bar{t}_{rj}$ в возвратное состояние S_j есть среднее время между двумя приходами в состояние S_j. Если приходы в S_j непериодичны, т. е. если времена возвращения t_{rj} не все одинаковы, S_j называется *апериодическим*. Апериодическая марковская цепь — это марковская цепь, все состояния которой апериодичны.

Обозначим через $p_j(k)$ вероятность того, что марковская цепь находится в состоянии S_j в момент времени k. Заметим, что p_j — *вероятности состояний*, тогда как p_{ij}, определенные выше,— *вероятности перехода*. Начальные вероятности состояний обозначим как $p_j(0)$.

Два важных результата позволят нам ответить на заданные выше вопросы.

Первое: если марковская цепь является однородной неприводимой и апериодической, то предельные вероятности состояний

$$p_j=\lim_{k\to\infty} p_j(k), \quad (j=0,\ 1,\ \ldots) \quad (4.17)$$

существуют и не зависят от начальных $p_j(0)$.

Второе: если все состояния марковской цепи являются возвратными и их средние времена возвращений конечны, то вероятности p_j образуют стационарное распределение и могут быть определены из уравнений:

$$p_j=\sum_i p_i p_{ij} \quad (j=0,\ 1,\ \ldots) \quad (4.18)$$

и

$$\sum_i p_i = 1. \quad (4.19)$$

Распределение вероятностей состояний называется *стационарным* в данном контексте, если, когда оно выбрано в качестве начального распределения, верно, что распределение вероятностей состояний марковской цепи для любого момента времени совпадает с начальным. Таким образом, если для всех j мы положим $p_j(0)=p_j$, то имеем $p_j(k)=p_j$ для всех k.

Решения уравнений (4.18) и (4.19) могут быть названы *равновесными вероятностями состояний*, поскольку они не зависят от начальных вероятностей состояний. При равновесии количество времени, которое система находится в состоянии S_j, пропорционально p_j. Кроме того, среднее время между двумя приходами в S_j, т. е. среднее время возвращения в S_j, может быть вычислено, если использовать другой полезный результат, который справедлив при тех же условиях, что и изложенный выше второй результат:

$$\bar{t}_{rj}=\frac{1}{p_j}. \quad (4.20)$$

Каково распределение временных интервалов t_j, проводимых системой в состоянии S_j? В каждый дискретный момент времени, если

марковская цепь находится в состоянии S_j, то с вероятностью p_{jj} она остается в состоянии S_j и с вероятностью $1-p_{jj}$ переходит в другое состояние. Вследствие марковского свойства (4.15) эти вероятности никогда не изменяются независимо от того, сколько времени уже было проведено в состоянии S_j. Независимость событий позволяет нам вероятность того, что длительность t_j в точности равна n моментам времени, вычислить как произведение вероятностей успешных событий, входящих в данное событие:

$$\text{Prob}\,(t_j = n) = (1 - p_{jj})\, p_{jj}^{n-1} \quad (n = 1, 2, 3, \ldots). \tag{4.21}$$

Таким образом, (4.21) показывает, что длительности состояний распределены геометрически со средним $1/(1-p_{jj})$. Важно отметить, что геометрическое распределение является единственным дискретным распределением, обладающим *свойством независимости от прошлого*. Если временные интервалы между некими событиями распределены геометрически, то в каждый момент времени интервал до следующего события статистически не зависит от того, сколько времени прошло с последнего события. То, что распределение длительностей состояний в марковской цепи с дискретным временем не зависит от прошлого, является прямым следствием марковского свойства (4.15). Аналогично можно показать, что в марковской цепи с непрерывным временем длительности состояний распределены экспоненциально и экспоненциальное распределение является единственным непрерывным распределением, для которого выполняется свойство независимости от прошлого.

Отсюда можно заключить, что в марковских моделях длительности состояний обязательно распределены или геометрически или экспоненциально. Для многих случаев это допущение не реально, даже если оно и сильно облегчает аналитическое решение модели. Это ограничение снимается в *полумарковских моделях*, в которых разрешены произвольные распределения для длительностей событий. Описание некой полумарковской модели для целей моделирования приведено в конце примера 3.3. Во время перехода в новое состояние эта модель ведет себя в точности как марковская модель. Поэтому говорится, что полумарковский процесс содержит *вложенный марковский процесс*, определенный в моментах переходов между состояниями. Конечно, множество марковских процессов является подмножеством множества полумарковских процессов.

Рассмотрим теперь простую марковскую модель, чтобы пояснить концепции и приемы, уже введенные в этом разделе.

Пример 4.3. Центральный процессор мультипрограммной системы в любой заданный момент времени выполняет команды из

a) программы пользователя или

b) подпрограммы операционной системы, явно вызываемой из программы пользователя и выполняющей для нее некую задачу (например, операцию по вводу-выводу); или

c) подпрограммы операционной системы, выполняющей общесистемную задачу управления, например планирование, распределение ресурсов или восстановление после сбоя системы; или

d) цикла ожидания.

Когда выполняемая команда принадлежит к категории a, центральный процессор находится в *состоянии задачи* (S_3). Когда выполняемая команда принадлежит к категории b или c, он находится в *состоянии супервизора*. Категория d соответствует *состоянию бездействия*. Предположим, что все подпрограммы операционной системы могут быть четко разделены по категориям b или c. Состояние, соответствующее категории b, назовем *состоянием супервизора для пользователя* (S_2), а соответствующее категории c — *состоянием супервизора для системы* (S_1). Такая классификация удобна для распределения расходов. Ясно, что нужно занести на счет каждого пользователя услуги, полученные им из категории b, и разделить услуги из категории c между всеми пользователями.

Наша модель центрального процессора имеет четыре состояния. Предположим, что любое состояние, если оно достигнуто, будет продолжаться не меньше чем 50 мкс,

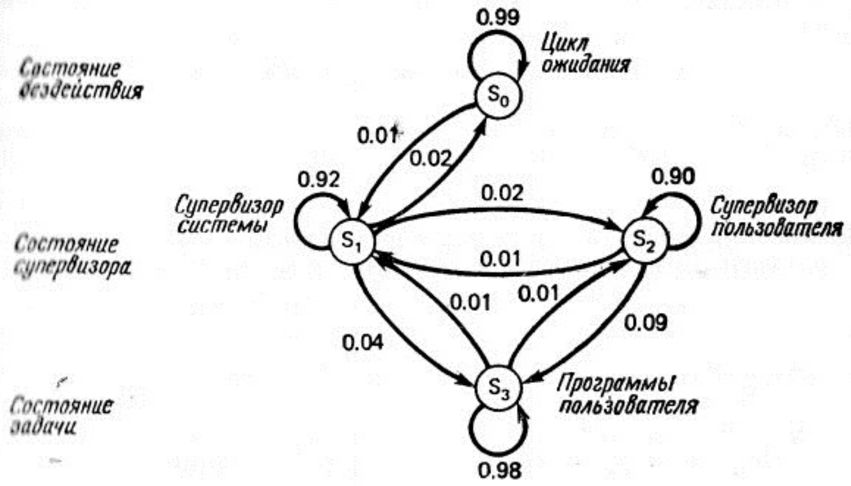


Рис. 4.5. Диаграмма переходов между состояниями для марковской модели центрального процессора с дискретным временем, описанной в примере 4.3.

и возьмем приблизительно продолжительности всех состояний такими, чтобы они были кратными 50 мкс. Тогда мы можем рассматривать марковскую модель центрального процессора с дискретным временем, взяв 50 мкс в качестве интервала между последовательными моментами дискретного времени.

Выбор марковской модели автоматически предполагает выполнение марковского свойства (4.15). Это значит, что если центральный процессор находится, например, в состоянии супервизора для системы, то вероятность перехода в состояние бездействия не зависит от того, было ли предыдущее состояние состоянием бездействия или состоянием задачи, или состоянием супервизора для пользователя. На самом деле, если предыдущее состояние было состоянием бездействия, то такой переход значительно менее вероятен, чем в остальных случаях. Тем не менее марковская модель не позволяет нам учесть такие отличия. Для того чтобы различать эти два случая, оставаясь в рамках марковских моделей, нам следовало бы разбить состояние супервизора для системы (S_1) на три состояния, характеризующиеся упорядоченными парами (S_3, S_1), (S_2, S_1) и (S_0, S_1). В этих парах первый элемент обозначает состояние, в котором система находилась непосредственно перед S_1. Таким образом, вероятности перехода из (S_2, S_1) в S_0 и из (S_0, S_1) в S_0 оказались бы различимыми и могли бы принимать разные значения.

Диаграмма переходов между состояниями для нашей однородной марковской модели изображена на рис. 4.5. В табл. 4.6 приведена матрица вероятностей переходов этой модели. Заметим, что до и после состояния бездействия всегда бывает состояние супервизора для системы. И в действительности планировщик центрального

процессора, когда процессор заканчивает выполнение задания и процессорная очередь пуста, переводит его в цикл ожидания. При этих условиях, когда обнаруживается прерывание, центральный процессор передает управление на супервизор.

Равновесные вероятности состояний вычисляются из системы уравнений (4.18) и (4.19):

$$\begin{aligned} p_0 &= 0.99p_0 + 0.02p_1, \\ p_1 &= 0.01p_0 + 0.92p_1 + 0.01p_2 + 0.01p_3, \\ p_2 &= \quad 0.02p_1 + 0.90p_2 + 0.01p_3, \\ p_3 &= \quad 0.04p_1 + 0.09p_2 + 0.98p_3, \\ 1 &= p_0 + p_1 + p_2 + p_3. \end{aligned} \tag{4.22}$$

Одно из первых четырех уравнений из (4.22) зависит от трех остальных. Поэтому для решения системы уравнение (4.19) должно рассматриваться вместе с (4.18). Решением (4.22) будет

$$p_0 = \frac{2}{9}, \quad p_1 = \frac{1}{9}, \quad p_2 = \frac{8}{99}, \quad p_3 = \frac{58}{99}. \tag{4.23}$$

Даваемый нашей моделью коэффициент использования центрального процессора составляет $p_1 = 1 - p_0 = 77.7\%$. Однако только приблизительно 58.6% всего времени

Таблица 4.6

Матрица вероятностей переходов, определяющая марковскую модель центрального процессора с дискретным временем из примера 4.3

		В состояние S_0	S_1	S_2	S_3
Из состояния	S_0	0.99	0.01	0	0
	S_1	0.02	0.92	0.02	0.04
	S_2	0	0.01	0.90	0.09
	S_3	0	0.01	0.01	0.98

в равновесном состоянии тратится процессором на обработку программ пользователя. Остаток времени 77.7—58.6=19.1% ЦП проводит в состоянии супервизора, 11.1% — в состоянии S_1 и 8% — в S_2.

Средняя продолжительность состояния S_j ($j=0, 1, 2, 3$) может быть вычислена из (4.21) как $50/(1-p_{jj})$ мкс. Таким образом, имеем $\bar{t}_0 = 5$ мс, $\bar{t}_1 = 625$ мкс, $\bar{t}_2 = 500$ мкс и $\bar{t}_3 = 2.5$ мс. Средние времена возвращения в эти четыре состояния определяются по (4.20) и должны быть умножены на 50 мкс:

$$\bar{t}_{r0} = 225_{\text{мкс}}, \quad \bar{t}_{r1} = 450_{\text{мкс}}, \quad \bar{t}_{r2} = 618.75_{\text{мкс}}, \quad \bar{t}_{r3} = 85.34_{\text{мкс}}. \tag{4.24}$$

Отметим, что эти времена очень малы, поскольку уход из текущего состояния всегда происходит в конце каждого интервала 50 мкс. Таким образом, когда следующее состояние совпадает с текущим, то происходит возврат в него, и мы имеем новый приход в это состояние со временем повторения, равным 1 интервалу.

Более интересные, чем (4.24), результаты можно получить при рассмотрении временного интервала между двумя последовательными переходами в S_j из некоторого другого состояния. Первые $1/(1-p_{jj})$ моментов этого интервала уходят в среднем на S_j, а остаток, длительность которого обозначим $\bar{t}'_{rj}$,— на другие состояния. Среднее время возвращения S_j можно вычислить следующим образом:

$$\bar{t}_{rj} = \frac{1 \cdot 1/(1-p_{jj}) + \bar{t}'_{rj} \cdot 1}{1 + 1/(1-p_{jj})}. \tag{4.25}$$

Таким образом, из (4.25) и (4.20) мы имеем

$$\bar{t}'_{rj} = \frac{2 - p_{jj} - p_j}{p_j (1 - p_{jj})}, \tag{4.26}$$

что дает

$$\begin{aligned} &\bar{t}'_{r0} = 17.7 \text{ мс}, \quad \bar{t}'_{r1} = 5.45 \text{ мс}, \quad \bar{t}'_{r2} = 6.3 \text{ мс}, \\ &\bar{t}'_{r3} = 1.85 \text{ мс}. \end{aligned} \tag{4.27}$$

■

Теперь мы готовы ввести *модели с очередями*. Основными компонентами моделей с очередями являются обслуживающие приборы, очереди и источники. *Обслуживающие приборы* в основном используются для моделирования ресурсов, запрашиваемых *заданиями*. Задания генерируются *источниками* или находятся в модели с очередями с момента их создания. Для каждого прибора число заданий, которые он может одновременно обслуживать, ограничено некоторым максимумом. Этот максимум часто называют *числом каналов* обслуживающего прибора. Те задания, которые поступают когда прибор занят, должны дожидаться в *очереди*, пока их не начнут обслуживать. Каждый прибор имеет по крайней мере одну очередь, а выражение *центр обслуживания* часто используется для обозначения комплекса, состоящего из обслуживающего прибора и его очередей. В некоторых случаях обслуживающий центр содержит несколько приборов, обслуживающих задания из общей очереди (очередей). Задание обычно требует внимания со стороны прибора на определенное количество времени (называемое *временем обслуживания*) и поступает в центр обслуживания в момент времени, называемый *временем поступления* задания в центр.

Важнейшими характеристиками источника являются:

a) его тип, *конечный* или *бесконечный*; если источник конечный, то максимальное число заданий, сгенерированных этим источником и находящихся в модели, имеет конечную верхнюю границу;

b) распределение интервалов между отдельными генерациями заданий *(времена между поступлениями)*;

c) *запросы* каждого задания на обслуживание для каждого центра обслуживания модели; если запросы на определенный тип обслуживания одинаково распределены для всех заданий, естественно рассматривать их скорее как характеристику центра обслуживания, чем источника; однако, поскольку они представляют запросы на ресурсы, вырабатываемые заданиями, то корректнее считать их характеристиками источника.

Центр обслуживания характеризуется:

a) числом и емкостью его очередей; *емкость* очереди — это максимальное число заданий, которые могут в ней содержаться;

b) числом обслуживающих приборов и числом каналов в каждом из них;

c) *скоростью* обслуживания; если задание требует d единиц обслуживания и v — скорость прибора, выраженная в представляемых им единицах обслуживания в единицу времени, то время обслуживания задания $t_s=d/v$; *средняя интенсивность обслуживания* для прибора за период времени τ определяется как $1/\bar{t}_s$ (где $\bar{t}_s$ — среднее время обслуживания в течение периода τ) и выражается числом заданий, обрабатываемых за единицу времени; когда скорость прибора фиксирована и запросы на обслуживание одинаково распределены для всех заданий или для каждого класса заданий, мы можем рассматривать распределение (или распределения) времен обслуживания как одну из характеристик прибора;

d) *дисциплина обслуживания*, которая определяет, при каких условиях приборы прекращают обслуживание задания, как выбирается из очередей центра обслуживания следующее задание, которое должно обслуживаться, и что делает не полностью обслуженное задание.

Модель с очередями определяется ее источниками, центрами обслуживания и их взаимосвязями. Взаимосвязи определяют пути,

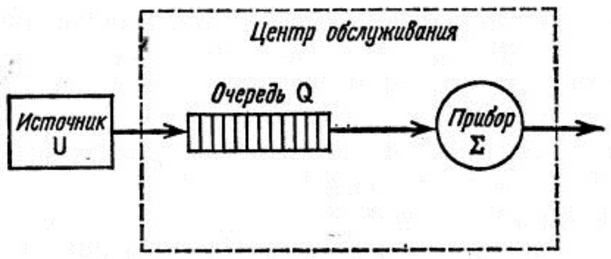


Рис. 4.6. Непрерываемая модель с очередями с одним центром обслуживания.

по которым заданиям разрешено следовать в модели от центра к центру.

Простейшим типом модели с очередями является модель с одним центром обслуживания (или с одним прибором), изображенная на рис. 4.6. Центр обслуживания состоит из одноканального прибора и одной очереди неограниченной емкости. Когда задание полностью обслужено прибором, оно покидает модель. Из заданий, находящихся в очереди, то задание, которое, согласно методу планирования центра, имеет наивысший по времени приоритет, поступает в прибор и обрабатывается им до завершения. Если очередь пуста, то прибор будет свободен до следующего поступления задания. Дисциплина обслуживания центра в этой модели должна быть *непрерываемой*. Иными словами, поскольку отстраненное прибором задание удаляется из системы, каждое задание должно обрабатываться до своего завершения. В моделях, в которых задание может заходить в один и тот же центр обслуживания несколько раз, обслуживание может быть *прерываемым*.

Модель, содержащая более одного обслуживающего центра, называется *сетью с очередями*. В сети с очередями одни центры обслужи-

вания работают как источники для других центров. Если, кроме того, существуют внешние источники, то сеть называется *открытой*. В противном случае сеть будет *замкнутой*. В замкнутой сети общее число заданий фиксировано и постоянно. Два простых примера открытой и замкнутой сетей представлены соответственно на рис. 4.7 и 4.8.

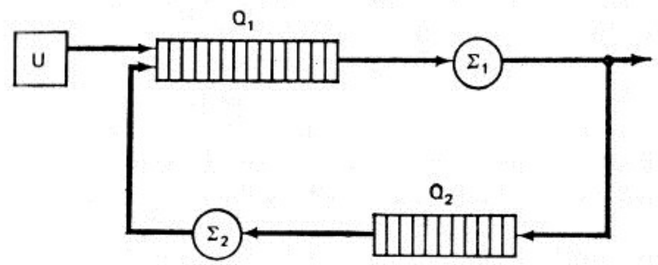


Рис. 4.7. Открытая сеть с очередями.

В модели с очередями для вычислительной системы, коллектив пользователей может быть представлен одним или более источниками заданий. Иначе он может моделироваться одним или несколькими обслуживающими центрами, которые пытаются воспроизвести поведение пользователей. Первый подход приводит к модели открытой сети, а

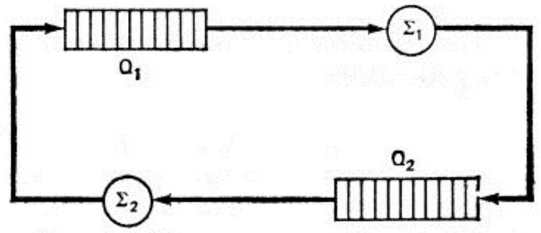


Рис. 4.8. Замкнутая сеть с очередями.

второй — к модели замкнутой сети. В то время как рабочая нагрузка в модели открытой сети представляется характеристиками источника (или источников), важнейшие из которых были перечислены выше, в замкнутой сети рабочая нагрузка моделируется характеристиками центра (или центров) обслуживания, представляющих коллектив пользователей.

Наиболее распространенные индексы производительности вычислительной системы (время ответа, время прохождения, пропускная способность, факторы использования) обычно легко определить, хотя и не всегда легко вычислить для моделей с очередями. Другие легко определяемые индексы — это времена ожидания в очередях и длины очередей. Конечно, для того чтобы можно было определить такие компонентно-ориентированные индексы, как коэффициент использования, времена ожидания и длины очередей, требуется, чтобы соответствующие компоненты были явно промоделированы в сети.

Не все ресурсы и не все особенности поведения заданий в вычислительных системах могут с такой же легкостью быть промоделированы в моделях с очередями. На самом деле в особенности трудно представлять ресурсы, которые в одном и том же задании используются взаимосвязанно с другими. Примером такого ресурса является основная память, часть которой должно получать задание, занимающее ее во время обслуживания или ожидания обслуживания центральным процессором или большинством устройств ввода-вывода. Другим примером является центральный процессор в системе, в которой выполнение задания перекрывается работами по вводу-выводу для этого же задания.

Требование аналитической разрешимости модели накладывает ряд дополнительных ограничений. Одно из них состоит во взаимной независимости времен между моментами появлений и временами обслуживания для всех приборов. Другие ограничения будут рассмотрены в следующем разделе.

Наше обсуждение моделей с очередями будет сосредоточено на моделях, предложенных для оценки вычислительных систем. Некоторые модели с очередями предназначались для исследования компонентов систем, таких, как диски, барабаны, буферы и устройства основной памяти. Читатель, которого интересуют такие модели, может обратиться к гл. 5 книги Коффмана и Деннинга (1973) и к библиографии к ней.

4.3.2. Непрерываемые модели с единственным центром обслуживания

Все обслуживающие приборы и каналы центра обслуживания извлекают задания, которые должны быть обработаны, из одной очереди или из одного множества очередей. Таким образом, в модели с единственным центром обслуживания может быть представлен только один тип ресурса. Этот ресурс не обязательно совпадает с одним каким-либо физическим ресурсом системы. Например, центр обслуживания может служить моделью всей вычислительной системы. Однако в подобной модели отсутствует способ представления различных ресурсов по отдельности. Модели с единственным центром обслуживания чаще всего применяются к изучению алгоритмов планирования, особенно для диалоговых систем.

В нашем рассуждении мы предположим, что модель на рис. 4.6 изображает некую диалоговую систему. Бесконечный источник U представляет собой коллектив пользователей. Сделанное предположение о бесконечности источника U становится все более оправданным по мере роста числа терминалов N.

Экспериментально обнаружено, что для некоторых диалоговых установок процесс поступления заданий достаточно близок к *пуассоновскому процессу* [см. Фукс и Джексон (1970), Андерсон и Сарджент (1972) и Льюис и Ю (1972)]. В пуассоновском процессе число n_a событий поступления в течение интервала τ имеет пуассоновское распределение:

$$\text{Prob}\,[n_a(\tau)=k]=\frac{(\lambda\tau)^k}{k!}e^{-\lambda\tau}\quad(k\geqslant 0,\ \tau\geqslant 0), \tag{4.28}$$

где λ — *средняя интенсивность поступления*. Легко может быть показано [см., например, Клейнрок (1975)], что пуассоновский процесс характеризуется экспоненциально распределенными временами между событиями поступления:

$$F_{t_{\text{int}}}(x)=\text{Prob}\,(t_{\text{int}}\leqslant x)=1-e^{-\lambda x}\quad(x\geqslant 0). \tag{4.29}$$

Как среднее, так и среднее квадратичное отклонение для показательного распределения равны $1/\lambda$. Отсюда *коэффициент вариации* c, являющийся отношением стандартного отклонения к среднему, равен 1.

Как отмечено в разд. 4.3.1, показательное распределение является единственным непрерывным распределением, обладающим свойством *независимости от прошлого*, что значительно облегчает анализ моделей с очередями. В терминах интервалов между событиями появления свойство независимости от прошлого означает, что во всякий момент времени интервал до следующего события поступления статистически не зависит от времени, которое уже прошло с момента предыдущего события поступления. Иначе говоря, вероятность поступления не зависит от истории процесса.

Если распределение времен обслуживания также предполагается показательным, то модель значительно легче решается аналитически, чем в случае какого-либо другого распределения времени обслуживания. К сожалению, экспериментально полученные распределения времен обслуживания центрального процессора имеют коэффициенты вариации, значительно превышающие 1, в основном от 3 до 10 [см. Андерсон и Сарджент (1972), Брайен (1967) и Парупуди и Виноград (1972)]. Мы вернемся к этому вопросу позднее.

Простейшая непрерываемая дисциплина обслуживания — это *первым пришел — первым обслужен* (FCFS) [1], называемая также *первым пришел — первым ушел* (FIFO), которая выбирает задания для обработки в хронологическом порядке поступления. Другие непрерываемые дисциплины, которые представляют интерес и были изучены, это дисциплина *самое короткое время обработки — первым* (SPTF), по которой выбираются задания с наименьшими запросами времени процессора (предполагается, что эти запросы известны), и дисциплина *внешний приоритет* (EP), в которой приоритет каждого задания определяется источником.

Рассмотрим теперь модель с очередями на рис. 4.6, в которой промежутки между поступлениями заданий и времена обслуживания независимы и одинаково распределены. Оба распределения предполага-

[1] Здесь и далее сокращенные обозначения получены из первых букв соответствующих английских терминов.— *Прим. ред.*

ются показательными. Средний промежуток между поступлениями обозначим $1/\lambda$, среднее время обслуживания — $1/\mu$. Таким образом, λ — средняя интенсивность поступлений, а μ — средняя производительность обслуживания. Эта модель часто обозначается символом $M/M/1$, где первая буква M означает, что промежутки между появлениями распределены экспоненциально (буква M от «марковский»),

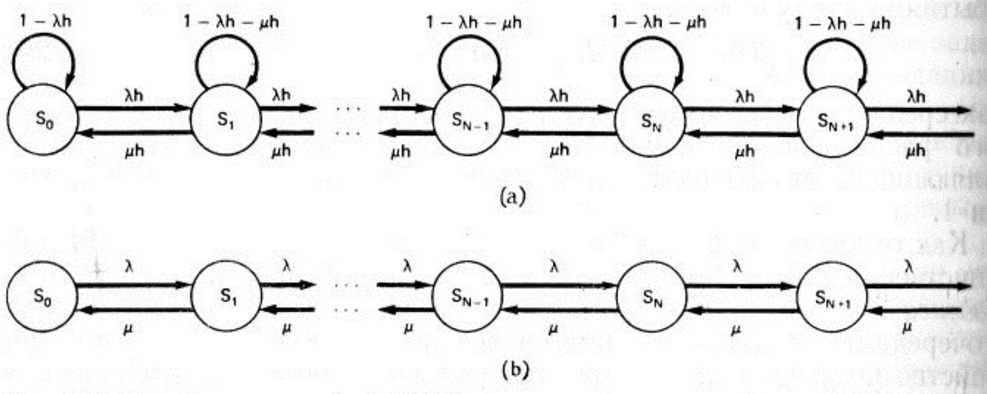


Рис. 4.9. Диаграммы состояний M/M/1-модели с очередями: (a)-диаграмма переходов состояний, (b)-диаграмма интенсивностей переходов состояний.

вторая M — что распределение времен экспоненциально, а 1 значит, что обслуживающий прибор имеет один канал.

Состояние модели с очередями с одним прибором, в которой все задания статистически идентичны, удобно описывается числом заданий N в центре обслуживания. N включает задания, ожидающие в очереди или очередях, а также обслуживаемые задания. Будем говорить, что модель находится в состоянии S_N ($N=0,1\ldots$), если она содержит N заданий.

Единственными событиями, которые могут изменить состояние модели, являются поступление и уход заданий. Мы предполагаем, что вероятностью того, что одновременно произойдут два таких события, можно пренебречь. Отсюда имеем диаграмму переходов состояний системы, показанную на рис. 4.9 (a). Сосредоточим внимание на интервале времени $(t, t+h)$, где h настолько мало, что вероятностью того, что в данный интервал времени произойдет более одного события, можно пренебречь. Если никакого события не произойдет, система не изменит своего состояния. Если произойдет событие поступления, то следующим состоянием будет соседнее состояние, изображенное справа от текущего на рис. 4.9 (a). Если произойдет уход, т. е. завершение обслуживания, система перейдет в соседнее состояние слева от текущего. Конечно, если текущее состояние S_0, то уходов быть не может. Вероятность появления в $(t, t+h)$ равна λh; она не зависит от истории процесса из-за свойства независимости от прошлого процесса появления. Точно так же, вероятность ухода в $(t, t+h)$ равна μh.

Исходя из наших предположений об экспоненциальности распределений, модель представляет собой однородную марковскую цепь с непрерывным временем. Такой тип марковской цепи, в котором переход из текущего состояния может произойти только в одно из ближайших соседних, называется *процессом рождения и гибели*. $M/M/1$-модель может рассматриваться как процесс рождения и гибели с постоянной *интенсивностью рождения* λ и постоянной *интенсивностью гибели* μ. Отметим, что *постоянный* в предыдущем предложении значит не только «постоянный во времени», но и «одинаковый для всех состояний». Можно показать, что предельные вероятности состояний существуют и стационарны, если $\lambda<\mu$.

Модель такого типа легче решить, чем произвольную марковскую модель с непрерывным временем. Вероятность $p_N(t+h)$ того, что система будет находиться в состоянии S_N в момент $t+h$, определяется суммой

a) условной вероятности того, что никаких событий в $(t, t+h)$ не произойдет, при условии, что в момент t система находилась в состоянии S_N: $(1-\lambda h-\mu h)p_N(t)$;

b) условной вероятности поступления в $(t, t+h)$ при условии, что в момент t система находилась в состоянии S_{N-1}: $\lambda h p_{N-1}(t)$;

c) условной вероятности ухода в $(t, t+h)$ при условии, что в момент t система находилась в состоянии S_{N+1}: $\mu h p_{N+1}(t)$.

Заметим, что если $N=0$, то условная вероятность для a) будет записана как $(1-\lambda h)p_0(t)$, а условная вероятность для b) равна нулю. Таким образом, имеем

$$p_0(t+h)=(1-\lambda h)\,p_0(t)+\mu h p_1(t), \tag{4.30}$$

$$p_N(t+h)=(1-\lambda h-\mu h)\,p_N(t)+\lambda h p_{N-1}(t)+\mu h p_{N+1}(t) \quad (N=1, 2, \ldots). \tag{4.31}$$

Вычитая $p_0(t)$ из обеих частей (4.30) и $p_N(t)$ из обеих частей (4.31), деля обе части на h и устремляя h к нулю, получаем

$$p_0'(t)=-\lambda p_0(t)+\mu p_1(t), \tag{4.32}$$

$$p_N'(t)=-(\lambda+\mu)\,p_N(t)+\lambda p_{N-1}(t)+\mu p_{N+1}(t) \quad (N=1, 2, \ldots). \tag{4.33}$$

Вспоминая, что предельное распределение вероятностей состояний стационарно, имеем $p_N'(t)=0$ для $t\to\infty$ и $N=0, 1\ldots$. Таким образом, в ситуации равновесия обыкновенные дифференциально-разностные уравнения (4.32) и (4.33) становятся линейными алгебраическими:

$$\lambda p_0=\mu p_1, \tag{4.34}$$

$$(\lambda+\mu)\,p_N=\lambda p_{N-1}+\mu p_{N+1} \quad (N=1, 2\ldots), \tag{4.35}$$

которые должны быть дополнены уравнением

$$\sum_{N=0}^{\infty} p_N=1. \tag{4.36}$$

Уравнения (4.34) и (4.35), называемые *равновесными уравнениями вероятностей состояний*, могут быть выписаны прямо из *диаграммы интенсивностей переходов состояний* нашей модели, представленной на рис. 4.9(b). В этой диаграмме метки ребер представляют интенсивности, а не вероятности. При равновесии интенсивности входного и выходного потоков для каждого состояния должны быть равны. Интенсивность входного потока в S_N равна $\lambda p_{N-1}+\mu p_{N+1}$. Интенсивность выходного потока из S_N равна $(\lambda+\mu)p_N$. Если мы потребуем равенства этих интенсивностей, то получим уравнение (4.35). Точно так же условия равновесия для S_0 дают уравнение (4.34).

Обозначим через ρ отношение λ/μ, которое предполагается меньшим 1. Это отношение называется в теории очередей интенсивностью нагрузки. Решением уравнений (4.34), (4.35) и (4.36) будет

$$p_0 = 1-\rho, \tag{4.37}$$

$$p_N+\rho^N p_0 = (1-\rho)\,\rho^N \quad (N=1, 2, \ldots). \tag{4.38}$$

Как видно из равенства (4.38), которое при $N=0$ сводится к (4.37), равновесные вероятности состояний для $M/M/1$-модели имеют геометрическое распределение. Математическое ожидание числа заданий в обслуживающем центре и его дисперсия могут быть выведены из (4.37) и (4.38) следующим образом:

$$\overline{N} = \sum_{N=0}^{\infty} N p_N = p_0 \sum_{N=1}^{\infty} N\rho^N = \frac{\rho}{1-\rho}, \tag{4.39}$$

$$\mathrm{Var}\,(N) = \sum_{N=0}^{\infty} (N-\overline{N})^2\, p_N = \frac{\rho}{(1-\rho)^2}. \tag{4.40}$$

Пользуясь такими же методами, убеждаемся, что вероятность того, что в центре обслуживания находится k или больше запросов, равна ρ^k.

Поскольку $p_0=1-\rho$ — это равновесная вероятность того, что прибор свободен, ρ есть равновесная вероятность того, что прибор занят, т. е. его коэффициент использования.

На этой модели могут быть легко определены и другие важные индексы производительности. Средняя пропускная способность совпадает с λ, средней интенсивностью входного потока, тогда, когда существует равновесное решение ($\rho<1$). И действительно, при равновесии задания не могут накапливаться в модели и средняя интенсивность выходного потока должна быть равна средней интенсивности входного потока. Отсюда максимальное значение средней пропускной способности равно μ, средней интенсивности обслуживания. Как было показано в конце разд. 4.3.1, эта интенсивность зависит от скорости обслуживающего прибора и от запросов заданий на обслуживание.

Среднее время ответа или время прохождения моделируемой системы соответствует среднему времени пребывания задания в модели. Для его вычисления мы применим результат Литтла: для любого распределения времени между двумя событиями поступления и любого распределения времени обслуживания, для любого числа каналов в центре обслуживания и для любой дисциплины обслуживания среднее число заданий в центре обслуживания равно произведению интенсивности поступления на среднее время пребывания в модели [Литтл (1961)]:

$$\overline{N} = \lambda \bar{t}. \tag{4.41}$$

Таким образом, из (4.41) и (4.39) получим

$$\bar{t} = \frac{1}{\mu\,(1-\rho)}. \tag{4.42}$$

Как для отдельного пользователя, так и для разработчика системы или ее администратора, часто бывает важно знать ожидаемое время ответа для задания с указанным запросом на время обслуживания. Пользователя, чье задание требует t_s секунд времени обслуживающего прибора, очевидно больше интересует ожидаемое время ответа для этого задания, чем значение $\bar{t}$, данное в (4.42), являющееся усредненным для всех заданий. Разработчик или администратор диалоговой системы обычно заинтересован в том, чтобы быстрее обрабатывались более короткие задания, поскольку это то, чего ждут от него пользователи.

Среднее $\overline{t(t_s)}$, рассматриваемое как функция от t_s, называется *функцией ответа*. Чтобы ее вычислить, мы должны знать стратегию планирования, применяемую в данном центре. Полученные до сих пор результаты не зависели от стратегии планирования. При стратегии FCFS среднее время ожидания $\bar{t}_w$ для задания Q равно среднему времени обслуживания, умноженному на среднее число заданий, находящихся в центре при поступлении этого задания. Таким образом, из (4.39) мы имеем

$$\overline{t(t_s)} = t_s + \bar{t}_w = t_s + \frac{1}{\mu}\cdot\frac{\rho}{1-\rho}, \tag{4.43}$$

что сводится к (4.42) для $t_s=1/\mu$.

Другим важным индексом для диалоговых систем является дисперсия времен ответа.

Еще более интересным индексом является дисперсия времен ответа для заданной команды или команд со сравнимым *весом*, т. е. для команд, чьи времена выполнения не очень отличаются. Эти индексы могут быть вычислены для $M/M/1$-модели, так как распределение времен ответа и времен ожидания могут быть выведены аналитически. Оба эти распределения экспоненциальные:

$$f_t(x) = \mu\,(1-\rho)\,e^{-\mu(1-\rho)x} \quad (x\geqslant 0), \tag{4.44}$$

$$f_{t_w}(y) = (1-\rho)\,\mathrm{imp}_0(y) + \lambda\,(1-\rho)\,e^{-\mu(1-\rho)y} \quad (y\geqslant 0;\ \mathrm{FCFS}), \tag{4.45}$$

где $\mathrm{imp}_0(y)$ — единичная импульсная функция устройства в начале ($y=0$). Таким образом мы имеем

$$\mathrm{Var}\,(t)=\frac{1}{\mu^2(1-\rho)^2}, \tag{4.46}$$

$$\mathrm{Var}\,(t_w)=\frac{\rho\,(2-\rho)}{\mu^2(1-\rho)^2}. \tag{4.47}$$

Заметим, что (4.47) может быть выведено непосредственно из (4.46) и соотношения

$$\mathrm{Var}\,(t)=\mathrm{Var}\,(t_s)+\mathrm{Var}\,(t_w)=\frac{1}{\mu^2}+\mathrm{Var}\,(t_w). \tag{4.48}$$

Это соотношение справедливо, так как два компонента t (t_s и t_w) статистически независимы, и поэтому $\mathrm{Var}\,(t)$ является суммой их дисперсий. Дисперсия времен ответа для данной команды равна величине $\mathrm{Var}\,(t_w)$, которая задана в (4.47). Отметим, что стандартное отклонение времен ответа, $\sigma_t=\sqrt{\mathrm{Var}\,(t)}$, совпадает с $\bar{t}$ и быстро возрастает при увеличении ρ. Отметим также, что это равенство не выполняется для распределения t_w, чье стандартное отклонение тоже очень резко возрастает при увеличении ρ.

Пример 4.4. В простой однопрограммной диалоговой системе команды и входные сообщения поступают от большого числа независимых терминалов. Когда наступает очередь для обработки некой команды или сообщения, соответствующий процесс загружается в основную память и под его управление передается центральный процессор, а также нужные ему устройства внешней памяти. Когда выполнение заканчивается, процесс разгружается из памяти и на соответствующий терминал посылается выходное сообщение. Следующее задание в очереди начинает обслуживаться только после того, как закончится операция разгрузки предыдущего. Отметим, что при этом отсутствует квантование времени, или, что то же самое, квант времени бесконечен.

Если мы предположим, что все входные запросы статистически идентичны и независимы, что процесс их поступления можно моделировать пуассоновским процессом и что их времена обслуживания не зависят от времен их поступления и распределены экспоненциально, то можем воспользоваться $M/M/1$-моделью с очередями. Отметим, что время обслуживания задания в данной модели не совпадает с его запросом времени центрального процессора. Оно является общим однопрограммным временем выполнения задания, включающим процессорное время и время ввода-вывода (но не время обмена с терминалом). На самом деле в такой системе центральный процессор простаивает до тех пор, пока выполняемый процесс ожидает окончания инициированной им операции ввода-вывода.

Таблица 4.7

Параметры $M/M/1$-модели с очередями из рис. 4.6

Название параметра	Обозначение	Значение
Средняя интенсивность появления	λ	1.5 заданий/с
Среднее время обслуживания (среднее однопрограммное время выполнения)	$1/\mu$	500 мс

Для значений параметров модели, приведенных в табл. 4.7, мы имеем $\rho=0.75$. Интенсивность нагрузки этой модели нельзя интерпретировать как коэффициент использования процессора, а нужно рассматривать как коэффициент использования комплекса процессора и устройств внешней памяти. Из (4.39) имеем $\bar{N}=3$, из (4.40) — $\mathrm{Var}\,(N)=12$, из (4.42) — $\bar{t}=2$с и из (4.46) — $\sigma_t=2$с. Среднее время ожидания для запроса в очереди будет по (4.43) $\bar{t}_w=\rho\bar{t}=1.5$с. Таким образом, для команды, имеющей в однопрограммном режиме время выполнения 4с, следует ожидать времени ответа 5.5с при стратегии планирования FCFS. Стандартное отклонение времени ответа по (4.47) будет 1.93с.

Кривая (а) на рис. 4.10 изображает среднее время ответа данной модели как функцию от ρ в предположении, что λ меняется от 0 до μ, а величина μ остается постоянной. Когда ρ меняется из-за изменения μ, а λ остается постоянной, мы получаем кривую (а′) на той же диаграмме. Уравнение этой кривой легко выводится из (4.42).

Наиболее заметной особенностью этих двух кривых является их резкое возрастание (до бесконечности) при приближении ρ к 1. Загрузка системы, при которой средняя пропускная способность λ близка к максимальной скорости μ делает $\bar{t}$ недопустимо большим. Конфликт между подходами администратора установки и пользователя не может быть изображен с большей ясностью, чем на этих диаграммах. Для относительно небольших значений ρ величина $\bar{t}(\rho)$ незначительно больше $1/\mu$ — среднего времени обслуживания. При $\rho=0.5$ среднее время ожидания $\bar{t}_w$ [см. (4.43)] равно $1/\mu$. При дальнейшем увеличении ρ значение $\bar{t}_w$ растет очень быстро и скоро начинает значительно превышать $1/\mu$. Для $\rho=0.9$ величина $\bar{t}_w$ больше $1/\mu$ в 9 раз.

Функция ответа модели в предположении, что стратегия планирования есть FCFS задается формулой (4.43) и изображена на рис. 4.11 [кривая (а)].

Как мы уже говорили, экспериментально было найдено, что предположение об экспоненциальном распределении является плохим приближением распределений времен обслуживания для компонентов вычислительной системы. Результаты измерений показывают, что коэффициенты вариации для распределений времен центрального процессора в основном значительно больше 1, а для распределения времени ввода-вывода — меньше 1.

Неэкспоненциальные распределения с $c>1$ можно аппроксимировать с помощью взвешенной суммы экспонент (*гиперэкспоненциальное распределение*) следующим образом:

$$F_{t_s}(x)=\mathrm{Prob}\,(t_s\leqslant x)=\sum_{i=1}^{k}w_i\,(1-e^{-\mu_i x})\quad(\mu_i>0,\ w_i>0,\ \sum_{i=1}^{k}w_i=1). \tag{4.49}$$

Математическим ожиданием и дисперсией случайной величины будут

$$\bar{t}_s=\sum_{i=1}^{k}\frac{w_i}{\mu_i},\quad \mathrm{Var}\,(t_s)=\sum_{i=1}^{k}\frac{2w_i}{\mu_i^2}-\left(\sum_{i=1}^{k}\frac{w_i}{\mu_i}\right)^2. \tag{4.50}$$

Легко показать, что коэффициент вариации

$$c=\frac{\sqrt{\mathrm{Var}\,(t_s)}}{\bar{t}_s}\geqslant 1 \tag{4.51}$$

и что $c=1$ только тогда, когда $\mu_i=\mu$ для всех i, т. е. когда распределение экспоненциально.

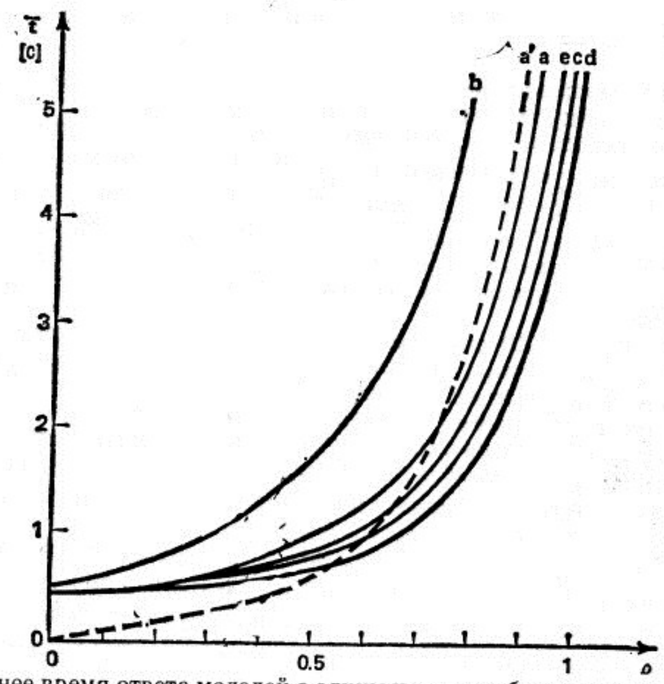


Рис. 4.10. Среднее время ответа моделей с одним центром обслуживания в зависимости от ρ: (a) — $M/M/1$-модель из примера 4.4. и PS-модель из примера 4.7, μ постоянно; (a′) — то же, что (a), λ постоянно; (b) — $M/G/1$-модель из примера 4.5, $c=2$, μ постоянно; (c) — $M/G/1$-модель из примера 4.5, $c=0.5$, μ постоянно; (d) — $M/D/1$-модель из примера 4.5; (e) — прерываемая RR-модель из примера 4.6, $q=0.3$ с.

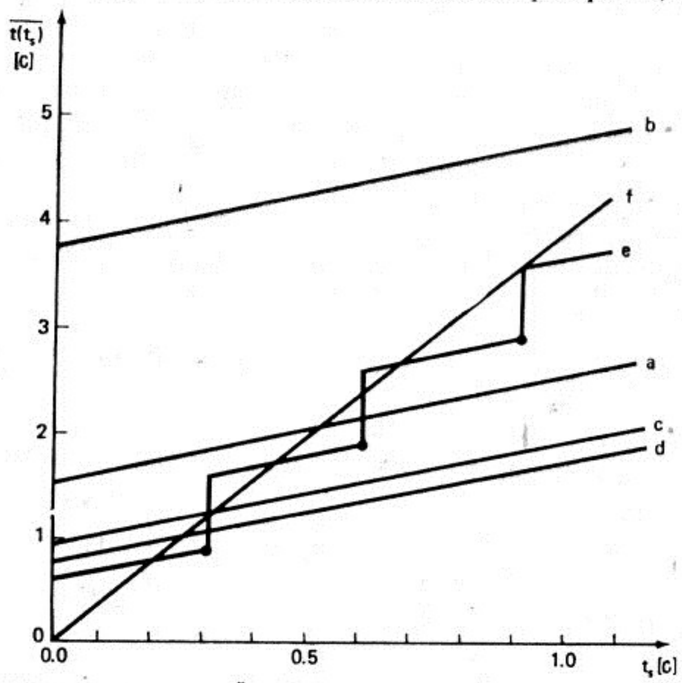


Рис. 4.11. Функции ответа моделей с одним центром обслуживания: (a) — $M/M/1$-модель с дисциплиной FCFS из примера 4.4; (b) — $M/G/1$-модель с дисциплиной FCFS из примера 4.5, $c=2$; (c) — $M/G/1$-модель из примера 4.5, $c=0.5$; (d) — $M/D/1$-модель из примера 4.5 с дисциплиной FCFS; (e) — прерываемая RR-модель из примера 4.6, $q=0.3$ с; (f) — PS-модель из примера 4.7.

Гиперэкспоненциальное распределение времен обслуживания, такое, как в (4.49), можно получить при параллельном соединении k экспоненциальных обслуживающих приборов. Прибор Σ_i должен иметь интенсивность обслуживания, равную μ_i, и вероятность быть использованным для обслуживания задания, равную w_i (рис. 4.12). Заметим, что в произвольный момент времени может быть занято не

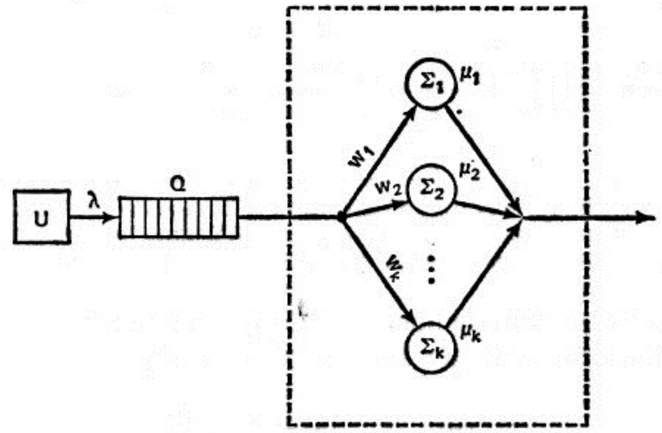


Рис. 4.12. Эквивалент модели с пуассоновским входным потоком и гиперэкспоненциальным распределением времен обслуживания с одним центром обслуживания из элементов с экспоненциальным распределением времен обслуживания.

более одного прибора из k. Для модели с одним обслуживающим центром, представленной на рис. 4.2, можно найти равновесное решение, определив состояние системы, как пару (N, i), где N — общее число заданий в центре, а i — идентификатор занятого прибора, если такой имеется ($i=0, 1 \ldots k$). При этом мы выписываем равновесные уравнения вероятностей состояний для неизвестных $p(N, i)$.

Аналогично, неэкспоненциальные распределения с $c<1$, такие, как у большинства времен обслуживания ввода-вывода, можно аппроксимировать *гипоэкспоненциальным распределением*, например следующего типа:

$$F_{t_s}(x) = \text{Prob}\,(t_s \leqslant x) = 1 - \frac{\mu_2}{\mu_2-\mu_1} e^{-\mu_1 x} - \frac{\mu_1}{\mu_1-\mu_2} e^{-\mu_2 x} \quad (\mu_1 \neq \mu_2). \tag{4.52}$$

Математическим ожиданием и дисперсией распределения (4.52) будут

$$\bar{t}_s = \frac{1}{\mu_1} + \frac{1}{\mu_2}, \quad \text{Var}\,(t_s) = \frac{1}{\mu_1^2} + \frac{1}{\mu_2^2}, \tag{4.53}$$

а коэффициент вариации имеет вид

$$c = \sqrt{1 - \frac{2\mu_1\mu_2}{(\mu_1+\mu_2)^2}} < 1. \tag{4.54}$$

Распределение времени обслуживания (4.52) можно получить последовательным соединением двух экспоненциальных приборов с ин-

тенсивностями обслуживания μ_1 и μ_2, причем во всякий момент времени должно быть занято не более одного прибора. Связывая последовательно k экспоненциальных приборов, мы можем моделировать гипоэкспоненциальное распределение более сложной математической формы, чем (4.52). Соответствующий многоприборный центр обслуживания изображен на рис. 4.13. Когда все приборы в последовательности

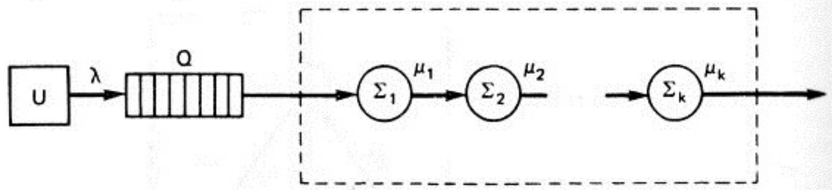


Рис. 4.13. Эквивалент модели с пуассоновским входным потоком и с одним центром обслуживания с гипоэкспоненциальным распределением времен обслуживания на элементах с экспоненциальным распределением времен обслуживания.

имеют одинаковую интенсивность обслуживания μ, распределением обслуживания центра будет *k-распределение Эрланга*

$$F_{t_s}(x) = 1 - e^{-k\mu x} \sum_{i=0}^{k-1} \frac{(k\mu x)^2}{i!}. \tag{4.55}$$

Модель на рис. 4.13 может быть решена так же, как модель с гиперэкспоненциальными временами обслуживания, если определить состояния системы как пары (N, i), где $i=0, 1, \ldots k$.

Гиперэкспоненциальные и гипоэкспоненциальные аппроксимации реальных распределений времен обслуживания бывают полезны в сетях с очередями, так как они позволяют получить сети, в которых все обслуживающие приборы экспоненциальны. Тем не менее приблизительное равенство коэффициентов вариации двух распределений является необходимым, но не достаточным условием для использования одного из них как приближения другого.

При изучении непрерываемых систем с одним центром обслуживания, имеющих произвольное распределение времен обслуживания (они называются также $M/G/1$-системы), нет необходимости в этих аппроксимациях, если нас интересует только индекс $\bar{t}$, так как при равновесности можно получить выражение для $\bar{t}$ в замкнутой форме. Если $1/\mu$ и c означают соответственно математическое ожидание и коэффициент вариации распределения времени обслуживания, то имеем

$$\bar{N} = \rho + \frac{\rho^2(1+c^2)}{2(1-\rho)}. \tag{4.56}$$

Это выражение для среднего числа заданий в центре обслуживания можно получить, если заметить, что число заданий, оставшихся в центре после ухода задания, является полумарковским процессом с вложенной марковской цепью, определенной в моменты ухода. Используя результат Литтла (4.41), получаем так называемую *формулу Хин-*

чина — Полачека

$$\bar{t} = \frac{1}{\mu}\left[1 + \frac{\rho(1+c^2)}{2(1-\rho)}\right] \tag{4.57}$$

и для стратегии FCFS

$$\bar{t}_w = \frac{\rho(1+c^2)}{2\mu(1-\rho)}. \tag{4.58}$$

Наиболее примечательным в этом результате является то, что среднее время ответа и среднее время ожидания зависят только от математического ожидания $1/\mu$ и стандартного отклонения c/μ распределения времен обслуживания. Все более высокие моменты этого распределения не влияют ни на $\bar{t}$, ни на $\bar{t}_w$.

Заметим, что при $c=1$ уравнения (4.56), (4.57) и (4.58) становятся соответственно уравнениями (4.39), (4.42) и выражением для $\bar{t}_w$, использованным в (4.43).

Пример 4.5. Рассмотрим снова систему, которую мы представляли $M/M/1$-моделью в примере 4.4. Снимем теперь предположение об экспоненциальности распределения времен обслуживания. Для произвольного распределения времен обслуживания нам известно математическое ожидание $1/\mu$ и стандартное отклонение σ_s. Появление заданий, как и раньше, моделируется пуассоновским процессом со средней интенсивностью λ. Значения $1/\mu$ и λ заданы в табл. 4.7.

Если $\sigma_s = 1$с, то $c=2$. Из (4.56) имеем $\bar{N}=6.375$, из (4.57) — $\bar{t}=4.25$ с и из (4.58) — $\bar{t}_w=3.75$ с. Таким образом, бо́льшая изменчивость времен обслуживания, соответствующая увеличению c с 1 до 2, приводит к увеличению $\bar{N}$ на 112%, на столько же процентов возрастает $\bar{t}$ и на 150% возрастает $\bar{t}_w$. Кривые (b) на рис. 4.10 и 4.11 изображают $\bar{t}(\rho)$ и $\overline{t(t_s)}$ для данной $M/G/1$-модели. Так же, как в (4.43), мы имеем $\overline{t(t_s)} = \bar{t}_s + \bar{t}_w$.

Обратные результаты получаются, когда времена обслуживания менее изменчивы, чем в случае $M/M/1$. Для $\sigma_s=0.25$ с имеем $c=0.5$; следовательно, $\bar{N}=2.15$, $\bar{t}=1.43$ с и $\bar{t}_w=0.93$ с. Таким образом, по сравнению с $M/M/1$-моделью $\bar{N}$ уменьшилось на 28%, на столько же процентов уменьшилось $\bar{t}$ и $\bar{t}_w$ уменьшилось на 38%. Кривые (c) на рис. 4.10 и 4.11 относятся к этому случаю.

Когда $c=0$, то нет изменчивости времен обслуживания. Все задания имеют одно и то же время обслуживания $1/\mu$. Это $M/D/1$-модель, где буква D означает *детерминированная*. Для значений λ и μ в табл. 4.7 имеем $\bar{N}=1.87$, $\bar{t}=1.25$ с и $\bar{t}_w=0.75$ с. Функция $\bar{t}(\rho)$ для переменной λ и $\overline{t(t_s)}$ для планирования FCFS изображены соответственно на рис. 4.10 и 4.11 [кривые (d)]. ■

4.3.3. Прерываемые модели с одним центром обслуживания

Простейшая прерываемая модель с очередью изображена на рис. 4.14. Задание, покидающее обслуживающий прибор, не обязательно уходит из центра обслуживания. Существует ненулевая вероятность b того, что задание вернется в очередь и будет ждать дополнительного обслуживания. Задание может покинуть прибор по различным причинам, которые зависят от дисциплины обслуживания, принятой в обслуживающем центре.

Наиболее распространенной прерываемой дисциплиной обслуживания является *круговая* (RR), разрешающая прибору обрабатывать задание не дольше заданного *кванта* времени q. Если время обслуживания, необходимое заданию, превосходит q, то обработка задания

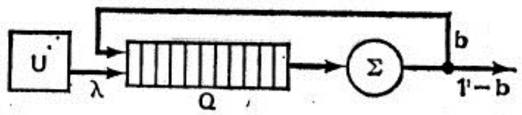


Рис. 4.14. Прерываемая модель с одним центром обслуживания.

прерывается и оно возвращается в конец очереди, управляемой дисциплиной FCFS. Дисциплина RR вследствие ее простоты, явилась одной из первых дисциплин, принятых в системах с разделением времени, и поэтому ей уделяется большое внимание [см. Мак-Кинни (1969)].

В вычислительных системах с дисциплиной RR вероятность b непостоянна. Например, для некоего конкретного задания значение b уменьшается, если это задание получает больше обслуживания. Однако требованиям разрешимости больше отвечает предположение о постоянности b. Как будет показано, даже если мы сделаем это допущение, а также дополнительное, о выполнении свойства независимости от прошлого как для распределения интервалов между поступлениями, так и для распределения времен обслуживания, решение этой модели является значительно более сложным, чем решение $M/M/1$-модели. Такая возросшая сложность проявляется всякий раз, когда дисциплина обслуживания зависит от запросов на обслуживание каждого задания и от пути, по которому каждое задание следует через модель.

Для простоты предположим, что времена обслуживания делятся нацело на квант q. Таким образом, чтобы сохранить свойство независимости от прошлого, мы предполагаем, что вероятность того, что задание требует k квантов, задается геометрическим распределением

$$\text{Prob}\,(t_s = kq) = b^{k-1}(1-b) \quad (k = 1, 2, \ldots; 0 < b < 1). \tag{4.59}$$

Положим, что состояние системы представляется числом заданий N в системе, и вероятность состояния N обозначим p_N $(N=0, 1, \ldots)$. Пусть задание, требующее k квантов обслуживания при своем поступлении, в условиях равновесия находит j заданий в центре обслуживания. Это задание, которое будет называться *возвращающимся заданием*, будет возвращаться назад в конце каждого из $k-1$ квантов полученного обслуживания. Таким образом, оно возвращается в очередь $k-1$ раз и уходит после получения k-го кванта. Шаг возвращающегося задания определяется как период, начинающийся, когда задание поступает в очередь и начинает ждать следующего кванта обслуживания, и кончающийся, когда задание покидает обслуживающий прибор после получения этого кванта. Время выполнения i-го шага обозначим τ_i $(i=1, \ldots, k)$.

В течение первого шага впереди возвращающегося задания находятся j заданий. Одно из этих заданий обслуживается в момент появления возвращающегося задания и оставит обслуживающий прибор через αq единиц времени $(0 \leqslant \alpha \leqslant 1)$. Мы можем, таким образом, написать

$$\tau_1 = \alpha q + (j-1)q + q. \tag{4.60}$$

Длительности последовательных шагов делятся нацело на q. Если h_i — это длительность i-го шага, выраженная в количестве квантов, то возвращающееся задание на этом шаге имеет перед собой $h_i - 1$ заданий, из которых в среднем $b(h_i-1)$ вернутся в очередь. Заметим, что этот результат основывается на предположении, что вероятность того, что задание потребует еще квант, не зависит от прошлого этого задания и, следовательно, задается как вероятность $\text{Prob}\,(t_s > q)$, которая по (4.59) равняется b. Это предположение основывается на свойстве независимости от прошлого, которым обладает распределение времен обслуживания.

В течение i-го шага возвращающегося задания произойдет в среднем $\lambda\bar{\tau}_i$ поступлений. Поскольку $h_i = \bar{\tau}_i/q$, то для $k \geqslant 3$ имеем

$$\bar{\tau}_{i+1} = b(h_i - 1)\,q + \lambda\bar{\tau}_i q + q = (\lambda q + b)\,\bar{\tau}_i + q\,(1-b) \quad (i = 2, \ldots, k-1) \tag{4.61}$$

или, иначе,

$$\bar{\tau}_{i+1} = a^{i-2}\bar{\tau}_2 + q\,(1-b)\,\frac{1-a^{i-2}}{1-a} \quad (i = 2, \ldots, k-1), \tag{4.62}$$

где

$$a = \lambda q + b. \tag{4.63}$$

Тот же подход, который мы использовали при написании (4.60), дает

$$\bar{\tau}_2 = bjq + \lambda\bar{\tau}_1 q + q = \lambda q\bar{\tau}_1 + q\,(1+bj). \tag{4.64}$$

Уравнения (4.64) и (4.62) позволяют нам определить среднее время, проводимое возвращающимся заданием в модели, если при его появлении было j заданий:

$$\bar{t}_s(j) = \sum_{i=1}^{k} \bar{\tau}_i = \bar{\tau}_1 + \frac{q\,(k-1)}{1-\rho} + q\,\frac{1-a^{k-1}}{1-a}\left(\lambda\bar{\tau}_1 + bj - \frac{\rho}{1-\rho}\right), \tag{4.65}$$

где

$$\rho = \frac{\lambda q}{1-b}. \tag{4.66}$$

Заметим, что ρ — коэффициент использования обслуживающего прибора, так как среднее время обслуживания по (4.59) равно $q/(1-b)$.

Средняя длина очереди определяется как

$$\sum_{N=1}^{\infty}(N-1)\,p_N = \sum_{N=1}^{\infty} N p_N - \sum_{N=1}^{\infty} p_N = \bar{N} - \rho. \tag{4.67}$$

Математическое ожидание αq равно $\rho q/2$, так как прибор загружен только в течение части всего времени, равной ρ. Заметив, что длина очереди при появлении возвращающегося задания равна $j-1$, мы можем вывести из (4.60) выражение для $\bar{\tau}_1$:

$$\bar{\tau}_1 = q\left(\bar{N} - \frac{\rho}{2} + 1\right). \tag{4.68}$$

Может быть показано, что значение $\bar{N}$ совпадает со значением N из уравнения (4.56), которое справедливо для $M/G/1$-модели:

$$\bar{N} = \rho + \frac{\rho^2(1+b)}{2(1-\rho)}. \tag{4.69}$$

Заметим, что коэффициент вариации распределения (4.59) равен $\sqrt{b}$. Из уравнения (4.65) выводится среднее время ответа для задания, требующего k квантов:

$$t_k = \sum_{j=0}^{\infty} p_j \bar{t}_k(j) = \bar{\tau}_1 + \frac{q(k-1)}{1-\rho} + q\frac{1-a^{k-1}}{1-a}\left(\lambda\bar{\tau}_1 + b\bar{N} - \frac{\rho}{1-\rho}\right)$$
$$(k = 1, 2 \ldots). \tag{4.70}$$

Используя уравнения (4.63), (4.66), (4.68) и (4.69), можно выразить величину $\bar{t}_k$ как функцию от параметров модели. Отметим, что так как из (4.63) и (4.66) мы получаем $a<1$, то при возрастании k зависимость $\bar{t}_k$ от времени обслуживания kq стремится к линейной [с коэффициентом $1/(1-\rho)>1$)].

Уравнение (4.70) может рассматриваться, как функция ответа модели. Среднее время ответа может быть вычислено [см. (4.59)] как

$$\bar{t} = \sum_{k=1}^{\infty} b^{k-1}(1-b)\,\bar{t}_k. \tag{4.71}$$

Пример 4.6. Моделируется однопрограммная диалоговая система, аналогичная рассматриваемой в примерах 4.4 и 4.5, но с круговой дисциплиной и конечным квантом. В этой системе задание при выполнении передает управление центральному процессору, когда выполняется одно из двух условий. Первым условием является прерывание задания, которое происходит обычно, когда требуется дополнительный ввод с терминала пользователя. Второе условие имеет место, когда данное задание управляло центральным процессором в течение полного кванта времени, начиная с момента, когда процессор в последний раз был назначен этому заданию, и имеется по крайней мере одно задание, ожидающее процессор. В прерываемой системе, такой, какая моделируется в примерах 4.4 и 4.5, центральный процессор освобождается, только когда имеет место первое условие. Поэтому такая система может рассматриваться как прерываемая система упомянутого типа, в которой квант времени бесконечен, так что второе условие никогда не выполняется.

Когда задание ожидает завершения инициированной им операции ввода-вывода, процессор не переключается на новое задание, а остается за этим заданием. Поэтому квант времени определяет максимум реального времени, а не полезного времени процессора, которое задание может использовать перед тем, как вернуться в очередь. Это реальное время, прошедшее с самого последнего назначения процессора исполняемому заданию.

Система, которую мы будем моделировать, такая же, как на рис. 4.14. Предположим, что средняя интенсивность входного потока равна 1.5 $с^{-1}$. Выберем квант времени $q=300$ мс и вероятность возвращения в очередь $b=0.4$. Для этих значений параметров среднее время обслуживания будет 500 мс, и из (4.66) $\rho=0.75$. Таким образом, мы имеем те же параметры, что и в примерах 4.4 и 4.5 (см. табл. 4.7).

Оценка (4.70) позволяет нам получить ординаты жирных точек кривой (е) на рис. 4.11. Функцию ответа нашей модели можно изобразить, если заметить, что внутри каждого интервала длины q от t_s значение $\bar{t}$ возрастает линейно с единичным наклоном.

Отметим, что ожидаемое время ответа на команду, время выполнения которой менее 300 мс, находится между 0.58 и 0.88. Для дисциплины FCFS кривая (а) рис. 4.11 дает ожидаемое время ответа между 1.5 и 1.8 с для того же диапазона затрат на время обслуживания. Сравнение функции ответа (а) и (е) подтверждает интуитивное предположение, что система с разделением времени дает лучшую реакцию для «быстрых» заданий, чем система, в которой всем заданиям разрешена работа до завершения, без обязанности освобождать центральный процессор. В случае на рис. 4.11 задания, запросы которых меньше 2 квантов обслуживания ($t_s<0.6$ с), получает в среднем более быстрый ответ от RR-системы с $q=0.3$ с, чем от FCFS-системы. Более длинные задания в среднем получают более быстрый ответ от FCFS-системы.

Из (4.71) имеем $\bar{t}=1.55$ с, что существенно меньше 2 с среднего времени ответа FCFS-системы, рассматриваемой в примере 4.4. Изменяя λ, мы можем изменить значение ρ, не изменяя при этом среднего времени обслуживания. Соответствующие значения $\bar{t}$, полученные из (4.71), изображены на рис. 4.10 [кривая (е)]. Эта кривая расположена ниже кривой, соответствующей FCFS-обслуживанию для всех значений ρ. ■

Если в модели на рис. 4.14 квант q стремится к 0, то это будет так, как если бы обслуживающий прибор во всякий момент времени одновременно обрабатывал все задания в обслуживающем центре, каждое задание со скоростью, обратно пропорциональной общему числу заданий. Такой метод распределения называется методом *разделения процессора* (PS).

Для пуассоновского потока и для любого распределения времени обслуживания со средним $1/\mu$ среднее время реакции для PS-модели задается формулой (4.42).

Для доказательства правильности этого утверждения вычислим сначала ожидаемое время ответа для задания, чей запрос времени обслуживания равен t_s. По (4.66) и (4.63) b и a должны стремиться к 1, когда q стремится к 0, а ρ остается постоянной. По (4.69) $\bar{N}$ будет стремиться к $\rho/(1-\rho)$, и по (4.68) $\bar{\tau}_1$ будет стремиться к 0. Если в (4.70) мы заменим k на t_s/q и устремим q к 0, то получим функцию ответа для PS-модели:

$$\overline{t(t_s)} = \frac{t_s}{1-\rho}. \tag{4.72}$$

Если времена обслуживания имеют распределение с функцией плотности f_{t_s}, мы можем написать

$$\bar{t} = \int_0^{\infty} f_{t_s}(x)\,\overline{t(x)}\,dx = \frac{\bar{t}_s}{1-\rho} = \frac{1}{\mu(1-\rho)}. \tag{4.73}$$

Так как при выводе формулы (4.73), которая идентична (4.42), не было сделано никаких предположений относительно распределения времен обслуживания, то можно заключить, что в PS-модели для любого распределения t_s величина $\bar{t}$ совпадает со средним временем ответа $M/M/1$-модели, имеющей те же характеристики λ и μ.

Пример 4.7. Если в системе с разделением времени, моделируемой в примере 4.6, квант времени будет выбран очень маленьким в сравнении со средним временем обслуживания, то мы можем получить приближение этой системы с помощью PS-модели с одним центром обслуживания.

В PS-системе все обрабатываемые задания должны находиться в основной памяти. Таким образом, система является мультипрограммной, даже если в модели с одним центром обслуживания явно не представлены устройства ввода-вывода. Как и в примерах 4.4—4.6, обслуживающий прибор в данной модели представляет не только центральный процессор, а комплекс всех процессоров в системе. Время обслуживания, требуемое заданием,— это не время центрального процессора, а общее однопрограммное время выполнения этого задания, включающее начальную загрузку задания в память и его конечную разгрузку из памяти.

Для $\lambda=1.5$ с$^{-1}$ и $1/\mu=500$ мс, как в табл. 4.7, ρ будет равно 0.75, и из (4.73) $\bar{t}=2$ с. Диаграмма $\bar{t}(\rho)$ будет совпадать с той, которую мы изобразили в примере 4.4 для $M/M/1$-модели [кривая (a) на рис. 4.10].

Функция ответа (4.72) для PS-модели приводится на рис. 4.11 [кривая (f)]. Это прямая линия, выходящая из начала координат под наклоном $1/(1-\rho)$. Для очень коротких заданий PS-модель выдает ответ быстрее, чем FCFS $M/M/1$-модель и даже чем RR-модель. Однако для $t_s>0.9$ с все команды обрабатываются быстрее при дисциплине RR, и то же самое происходит для некоторых диапазонов t_s при $t_s<0.9$ с. Кроме того, FCFS $M/M/1$-модель выдает ответ быстрее при $t_s>0.5$ с.

В реальных системах с разделением времени PS-метод и даже круговой метод с очень коротким квантом не может быть эффективно применен. При убывании q накладные расходы на переключение центрального процессора становятся относительно более значительными. В некий момент этими накладными расходами уже нельзя пренебрегать, поскольку процессор тратит большую часть своего времени на переключение с одного процесса на другой. Таким образом, PS-модели могут рассматриваться только как грубое приближение RR-системы, квант которой сравнительно мал относительно среднего, или, лучше, медианы времен обслуживания. ■

Прерывания обрабатываемого задания могут также произойти из-за поступления другого задания, имеющего более высокий приоритет. В этом случае решение о выборе задания для обработки должно приниматься при каждом поступлении задания. Некоторые дисциплины прерывания такого типа были исследованы для модели с одним центром обслуживания, изображенной на рис. 4.14, например дисциплина PSETF *(задание с наименьшим прошедшим временем — первым)*, которая удобна, когда времена обслуживания заданий заранее не известны, или дисциплина PSPTF *(задание с наименьшим временем обслуживания — первым)*, или PEP *(дисциплина с внешними приоритетами)*.

Если прерванные задания присоединяются к очереди, отличной от той, из которой они поступали на обслуживание, а прибор имеет всего n очередей, то мы получаем n-уровневую дисциплину *передний план — фон* (FB_n). FB_n-модель изображена на рис. 4.15. Эта модель, которая более точно представляет некоторые системы с разделением

времени, чем модель с одной очередью на рис. 4.14, может исследоваться при различных допущениях. В одной из версий с каждой очередью связан свой квант времени. Обычно $q_i<q_{i+1}$ для всех i, а q_n может быть бесконечным. Задание из очереди Q_i обслуживается, если очереди $Q_1, \ldots$

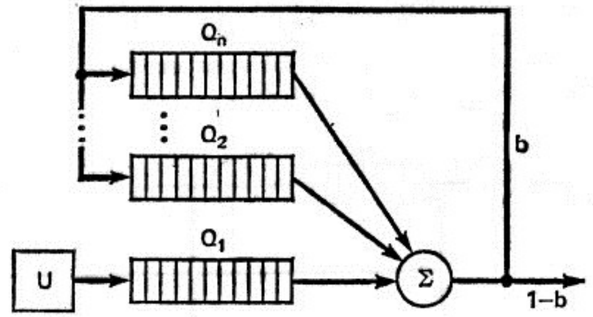


Рис. 4.15. Модель с очередями с одним центром обслуживания при дисциплине FB_n.

$\ldots, Q_{i-1}$ пусты. Прерывание происходит только по истечении кванта. Прерванное задание из очереди Q_i переходит в Q_{i+1}, а из Q_n возвращается в Q_n. Задания в каждой очереди обрабатываются в порядке FCFS.

4.3.4. Модели — открытые сети

При моделировании вычислительных систем открытые сети могут дать большую степень детализации, чем модели, содержащие только один центр обслуживания. Открытая сеть была определена в разд. 4.3.1 как сеть, имеющая по крайней мере один источник.

Работа Джексона (1963) позволяет нам решать модели — открытые сети, в которых распределение времен между событиями поступления и все распределения времени обслуживания экспоненциальны, а дисциплины обслуживания в центрах не зависят от будущих требований на время обслуживания и от дальнейшего пути задания через сеть. Последнее предположение исключает квантовые дисциплины и большинство других прерываемых дисциплин. Это является приемлемым ограничением для центров обслуживания, моделирующих, например, диск или барабан, но не для центров, представляющих центральный процессор в моделях диалоговых систем. Проиллюстрируем теперь подход Джексона на примере.

Пример 4.8. Используя метод, предложенный Джексоном, мы построим и решим аналитически модель ПРОП-системы, изображенной на рис. 1.2. Каждый из четырех процессоров P_1, P_2, P_3 и P_4 будет представлен как одноканальный центр обслуживания с одним обслуживающим прибором. Каждый центр будет иметь одну очередь неограниченной длины. Для того чтобы этот метод был применим, мы должны предположить, что времена обслуживания заданий в каждом центре независимы и экспоненциально распределены. Для i-го центра среднюю интенсивность обслуживания обозначаем через μ_i. Предполагается, что дисциплина обслуживания в каждом центре непрерываемая.

Коллектив пользователей представлен бесконечным источником, генерирующим задания согласно пуассоновскому процессу с интенсивностью λ.

Результаты Джексона позволяют обрабатывать модель, когда λ является функцией от числа заданий в модели N и когда каждая интенсивность обслуживания μ_i является функцией от N_i, где N_i — неотрицательное целое, представляющее число обслуживаемых или ожидающих заданий в i-м центре обслуживания. Для простоты, однако, предположим, что как λ, так и μ_i — константы. Заметим, что это допущение вполне приемлемо для значений μ_i тех центров, которые представляют компоненты с действительно непрерывной дисциплиной. В общем случае это допущение не точно для значений μ_i других центров и для λ, которое, особенно в сильно загруженных диалоговых системах, конечно, зависит от N.

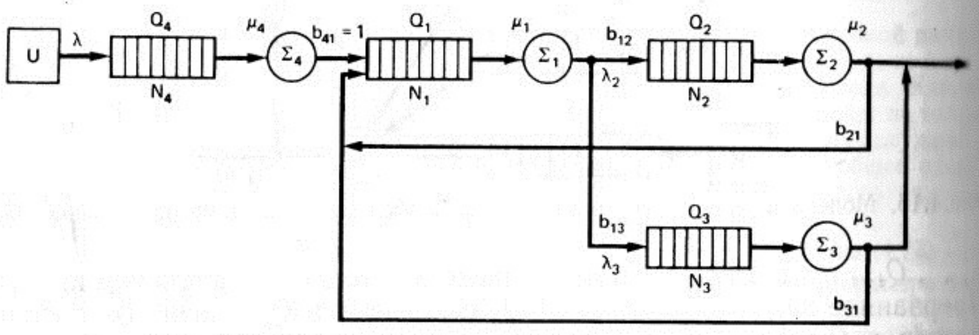


Рис. 4.16. Открытая сеть для ПРОП-системы с рис. 1.2.

Взаимосвязи между четырьмя центрами обслуживания отражают путь, по которому следует задание внутри системы. Как показано на рис. 4.16, задание входит в систему через процессор ввода-вывода P_3. Время, проведенное заданием в центре обслуживания 4, должно как-то отражать его операции ввода и загрузки. Задание, получившее запрашиваемое обслуживание в Σ_4, рассматривается как загруженное в основную память и готовое к выполнению. Когда приходит его очередь на обработку центральным процессором, то оно обслуживается прибором Σ_1. Необходимость в операции ввода-вывода заставляет задание покинуть центральный процессор, уступив место следующему заданию из Q_1, и перейти или в центр обслуживания 2, который является моделью процессора барабана, или в центр 3, моделирующий дисковый процессор. Вероятность запроса барабана, обозначенная b_{12}, предполагается одинаковой для всех заданий и постоянной по времени. Вероятность запроса диска b_{13} равна, естественно, $1-b_{12}$.

Когда обслуживающий прибор, соответствующий диску или барабану, завершает обработку запроса из задания, задание может завершиться и покинуть систему или вернуться в очередь центрального процессора Q_1 для дальнейшей обработки прибором центрального процессора. Вероятности возвращения в Q_1 обозначены соответственно b_{21} и b_{31}. Как и b_{12}, и b_{13}, эти вероятности одинаковы для всех заданий и постоянны. Заметим, что находящееся задание по завершении должно пройти через процессор P_4, чтобы его вывод мог достичь внешней среды за пределами системы. Эта операция вывода не представлена в модели на рис. 4.16.

Важным системным ресурсом, который тоже не моделируется, является основная память. Память трудно представить в модели с очередями. Это обусловлено тем, что она разделяется между заданиями и часто используется вместе с остальными ресурсами. Отметим, что все задания в центрах обслуживания 1, 2 и 3 будут рассматриваться как модели реальных заданий, которые должны сосуществовать в основной памяти. Предположение о бесконечной длине очередей в нашей модели с этой точки зрения, конечно, является необоснованным. В общем случае, для того чтобы каким-то образом представить емкость памяти, мы должны будем ограничить размеры очередей. Однако это значительно увеличивает сложность получения аналитического решения. Другой подход состоит в том, чтобы сохранить постоянным общее число заданий в центрах обслуживания 1, 2 и 3, т. е. сохранить постоянной степень мультипрограммирования. Этот подход описан в разд. 4.3.5.

В табл. 4.8 перечислены параметры модели, которую мы сконструировали для нашей системы. В этой модели, как и во всех открытых сетях, средняя пропускная способность равна интенсивности λ, которая является одним из параметров модели. Это, однако, не верно для среднего времени ответа или среднего времени прохождения, определение которых будет главной целью нашего анализа этой модели. Другим интересным индексом является максимальная пропускная способность, которую может

Таблица 4.8

Параметры модели — открытой сети на рис. 4.16

Название параметра	Обозначение	Значение
Средняя интенсивность появлений	λ	0.7 заданий/с
Среднее непрерывное время ЦП	$1/\mu_1$	30 мс
Среднее время обслуживания барабана	$1/\mu_2$	20 мс
Среднее время обслуживания диска	$1/\mu_3$	80 мс
Среднее время обслуживания ввода	$1/\mu_4$	500 мс
Вероятность запроса барабана	b_{12}	0.75
Вероятность запроса ЦП после обслуживания барабана	b_{21}	1
Вероятность запроса ЦП после обслуживания диска	b_{31}	0.9

обеспечить система при обработке рабочей нагрузки, состоящей из статистически одинаковых заданий с заданными значениями μ_i и b_{ij}.

Состояние модели на рис. 4.16 определяется как четырехмерный вектор $\mathbf{N}=(N_1, N_2, N_3, N_4)$. Соответствующее общее число заданий в системе равно $N=\sum_{i=1}^{4} N_i$. События в модели являются все поступления заданий и все завершения обслуживания задания в центре обслуживания.

Пусть система в момент времени t находится в состоянии $\mathbf{K}=(K_1, K_2, K_3, K_4)$. Вычислим теперь условные вероятности $p(\mathbf{N}, t+h|\mathbf{K}, t)$ того, что система будет находиться в состоянии $\mathbf{N}=(N_1, N_2, N_3, N_4)$ в момент времени $t+h$, где h настолько мало, что за интервал времени $(t, t+h)$ не может произойти более одного события. Вероятность появления в этом интервале времени равна $h\lambda$, а вероятность завершения обработки задания i-м центром обслуживания равна $h\mu_i$. Должны быть рассмотрены следующие четыре случая.

a) $\mathbf{N}=\mathbf{K}$ (событий не произошло). Таким образом,

$$p(\mathbf{N}, t+h \mid \mathbf{K}, t) = 1-h\lambda-h\sum_{i=1}^{4}\mu_i. \tag{4.74}$$

b) $\mathbf{N}=\mathbf{K}$, за исключением $N_4=K_4+1$ (одно появление). Отсюда

$$p(\mathbf{N}, t+h \mid \mathbf{K}, t) = h\lambda. \tag{4.75}$$

c) $\mathbf{N}=\mathbf{K}$, за исключением $N_2=K_2-1$ или $N_3=K_3-1$ (один уход). Таким образом,

$$p(\mathbf{N}, t+h \mid \mathbf{K}, t) = h\mu_x(1-b_{x1}), \tag{4.76}$$

где x равно 2 или 3.

d) $\mathbf{N}=\mathbf{K}$, за исключением $N_x=K_x-1$, $N_y=K_y+1$ (одно завершение обслуживания в центре x, сопровождающееся немедленным переходом в другой центр y). Для модели на рис. 4.16 имеем $x=4$ и $y=1$ или $x=1$ и $y=2$, или $x=1$ и $y=3$, или $x=2$ и

$y=1$, или $x=3$ и $y=1$. Заметим, что $b_{41}=1$. Таким образом,

$$p(\mathbf{N},t+h \mid \mathbf{K},t)=h\mu_x b_{xy}. \tag{4.77}$$

Из теории вероятности мы знаем, что

$$p(\mathbf{N},t+h)=\sum_{\text{все } \mathbf{K}} p(\mathbf{K},t)\, p(\mathbf{N},t+h \mid \mathbf{K},t). \tag{4.78}$$

Подставив (4.74), (4.75), (4.76) и (4.77) в (4.78), получаем

$$\begin{aligned} p(\mathbf{N},t+h)=p(\mathbf{N},t)&\left(1-h\lambda-h\sum_{i=1}^{4}\mu_i\right)+h\lambda p[(N_1,N_2,N_3,N_4-1),t]\\ &+h\mu_2(1-b_{21})\,p[(N_1,N_2+1,N_3,N_4),t]\\ &+h\mu_3(1-b_{31})\,p[(N_1,N_2,N_3+1,N_4),t]\\ &+h\mu_4 b_{41}p[(N_1-1,N_2,N_3,N_4+1),t]\\ &+h\mu_1 b_{12}p[(N_1+1,N_2-1,N_3,N_4),t]\\ &+h\mu_1 b_{13}p[(N_1+1,N_2,N_3-1,N_4),t]\\ &+h\mu_2 b_{21}p[(N_1-1,N_2+1,N_3,N_4),t]\\ &+h\mu_3 b_{31}p[(N_1-1,N_2,N_3+1,N_4),t]. \end{aligned} \tag{4.79}$$

Теперь вычтем $p(\mathbf{N}, t)$ из обеих частей (4.79), разделим обе части на h и устремим h к нулю. Мы получим дифференциально-разностное уравнение, выражающее $dp(\mathbf{N}, t)/dt$ как функцию от λ, μ_i, b_{ij} и вероятностей тех состояний, из которых $\mathbf{N}$ может быть достигнуто за один шаг. Джексоном (1963) было доказано, что решение такого уравнения существует и единственно.

Решение, которое является *распределением вероятностей равновесного состояния*

$$p(\mathbf{N})=\lim_{t\to\infty} p(\mathbf{N},\ t), \tag{4.80}$$

задается в виде

$$p(N_1,N_2,N_3,N_4)=p_1(N_1)\cdot p_2(N_2)\cdot p_3(N_3)\cdot p_4(N_4), \tag{4.81}$$

где

$$p_i(N_i)=(1-\rho_i)\,\rho_i^{N_i} \quad (i=1,\ 2,\ 3,\ 4), \tag{4.82}$$

причем

$$\rho_i=\frac{\lambda_i}{\mu_i}, \tag{4.83}$$

$$\begin{aligned} \lambda_1&=b_{21}\lambda_2+b_{31}\lambda_3+b_{41}\lambda_4,\\ \lambda_2&=b_{12}\lambda_1,\\ \lambda_3&=b_{13}\lambda_1,\\ \lambda_4&=\lambda. \end{aligned} \tag{4.84}$$

Заметим, что предполагается существование одного и только одного решения системы линейных алгебраических уравнений (4.84). Читатель может убедиться, что результаты в (4.81) и (4.82) удовлетворяют дифференциально-разностному уравнению, полученному из (4.79).

Таким образом, при наших допущениях равновесное распределение числа заданий в центре обслуживания i не зависит от распределения числа заданий в других центрах и является таким же, как в $M/M/1$-системе с очередями с интенсивностью появления λ и интенсивностью обслуживания μ_i [см. (4.38)].

Этот вывод позволяет нам использовать результаты, полученные для $M/M/1$-модели в разд. 4.3.2, для любой сети с очередями джексоновского типа. В частности, коэффициент использования i-го центра обслуживания просто равняется значению

задаваемому (4.83). Кроме того, в силу (4.39) среднее число заданий в центре обслуживания i будет

$$\overline{N}_i=\frac{\rho_i}{1-\rho_i}. \tag{4.85}$$

Из результата Литтла (4.41), среднее время ответа, определенное как среднее время между поступлением данного задания в систему и его уходом, равно

$$\overline{t}=\frac{1}{\lambda}\,\overline{N}=\frac{1}{\lambda}\sum_{i=1}^{4}\frac{\rho_i}{1-\rho_i}. \tag{4.86}$$

При значениях параметров из табл. 4.8 решение уравнений (4.84) дает $\lambda_1=28$ задание/с, $\lambda_2=21$ задание/с, $\lambda_3=7$ задание/с и $\lambda_4=0.7$ задание/с. Из (4.83) имеем $\rho_1=0.84$, $\rho_2=0.42$, $\rho_3=0.56$, $\rho_4=0.35$. Следовательно, среднее число заданий в системе $\overline{N}$ приблизительно равно 7.78, и по (4.86) $\overline{t}=11.1$ с. В среднем в центре обслуживания централь-

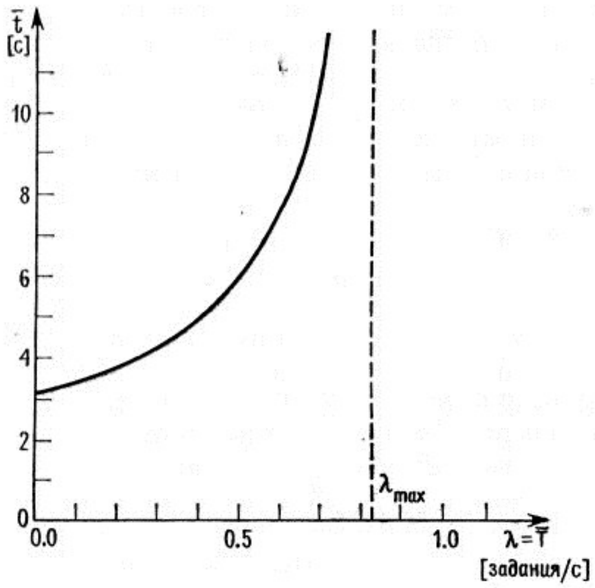


Рис. 4.17. Среднее время ответа в зависимости от средней пропускной способности модели рис. 4.16 и табл. 4.8.

ного процессора 5.25 задания, на барабане 0.72, на диске 1.27, и на процессоре ввода 0.54 задания.

Максимальная интенсивность поступления такова, что она полностью насыщает самый занятый центр, центральный процессор ($\rho_1=1$). Так как в нашей модели

$$\rho_1=\frac{1}{\mu_1}\,\lambda_1=\frac{1}{\mu_1}\,\frac{1}{1-b_{21}b_{12}-b_{31}b_{13}}\,\lambda=1.2\ \lambda, \tag{4.87}$$

то мы имеем

$$\overline{T}_{\max}=\lambda_{\max}=\frac{\rho_{1\max}}{1.2}=\frac{1}{1.2}=0.833 \text{ задание/с}. \tag{4.88}$$

График на рис. 4.17 показывает отношение между средним временем ответа и средней интенсивностью поступления, которая (для $\lambda\ll\lambda_{\max}$) совпадает со средней пропускной способностью $\overline{T}$. Заметим, что для $\lambda=0$ мы имеем $\overline{t}=3.1$ с. Это среднее время однопрограммного выполнения задания. Для $\lambda=0.7$ задание/с общее реальное

время, проведенное заданием в системе, приблизительно в 3.6 раза больше минимального значения $\bar{t}$. Замедление объясняется конкуренцией за ресурсы, увеличивающей среднее время ввода с 0.5 с до приблизительно 0.768 с, среднее общее время центрального процессора с 1.2 до 7.5 с, среднее общее время барабана с 0.6 до 1.034 с и среднее общее время диска с 0.8 с до 1.817 с. Эти результаты получены применением формулы Литтла для каждого отдельного центра обслуживания на рис. 4.17. ■

Баскетт и др. (1975) показали, что открытые и замкнутые сети с очередями и с центрами обслуживания, характеризующимися более общими распределениями времени обслуживания и дисциплинами выбора, включая FCFS, PS и LCFS (последним пришел — первым обслужен), могут быть решены в символьном виде. В этих сетях могут существовать различные классы заданий и заданию разрешается менять класс с заданной постоянной вероятностью при переходе из одного центра в другой. Каждый класс заданий в некоторых типах центров обслуживания может характеризоваться своим распределением времен обслуживания. Одна и та же сеть может быть открытой для одних классов заданий и замкнутой для других. Процесс поступлений может быть пуассоновским или состоять из конечного числа пуассоновских потоков.

Возможность символического решения сетей с очередями, содержащих различные классы заданий, позволяет при моделировании вычислительных систем значительно увеличить степень детализации моделей рабочей нагрузки. Рабочая нагрузка большинства установок может быть разбита на классы, определяемые существенно разными значениями характеристик задания (см. разд. 5.3.5). Очевидным, простым и очень распространенным примером является пример установки, обслуживающей пользователей, работающих как в диалоговом режиме, так и в пакетном. Запросы системных ресурсов в диалоговом режиме требуют по большей части совершенно других количеств системных ресурсов, чем те, которые требуются для пакета. Моделирование этой рабочей нагрузки в предположении, что распределения времен обслуживания для каждого центра и времен между поступлениями одинаковы для всех заданий, очевидно, является очень грубым приближением. При введении в модель различных классов заданий, корреляции между параметрами заданий учитываются естественным образом.

Когда неизвестно точное решение сетевой модели с очередями в замкнутой форме, то иногда оказывается возможным определить приближенное решение. Приближенные методы позволяют нам решать более сложные и, следовательно, потенциально более точные модели вычислительных систем. Ясно, что удобно использовать эти методы, только если вносимые ими погрешности менее значимы, чем та точность, которую дает детализация модели.

В частности, можно применить *диффузионное приближение* к открытым и замкнутым сетям с одноканальными центрами обслуживания с одним обслуживающим прибором, характеризуемыми произвольны-

ми распределениями и непрерывными тактиками планирования без использования априорной информации о времени обслуживания [см. Кобаяси (1974)]. Этот метод заключается в моделировании числа заданий в сети или в центре обслуживания стохастическим процессом с непрерывным, а не дискретным временем [см., например, Ньюэлл (1971)]. Функция плотности вероятности этого числа при соответствующих предположениях удовлетворяет диффузионному уравнению. Это уравнение может быть решено в некоторых случаях, когда исходный процесс с дискретным временем не вызывает непреодолимых трудностей. Иногда даже удается определить переходное уравнение модели.

Другой подход к приближенному решению сложных сетей с очередями основывается на идее *декомпозиции*. Как и в других областях, где используется декомпозиция, этот метод заключается в разбиении сети на подсети, в исследовании каждой подсети по отдельности, в замене ее приближенной моделью (например, одним центром обслуживания) и в анализе упрощенной сети. Таким образом, декомпозиция сетевой модели подобна ее трансформации в иерархию последовательно решаемых моделей. Метод декомпозиции сетей с очередями и критерии, которых нужно придерживаться при его применении, описаны у Куртуа (1971).

4.3.5. Модели — замкнутые сети

Типичное применение замкнутые сети с очередями могут найти при моделировании мультипрограммных систем, в которых степень мультипрограммирования N постоянна или поведение которых приходится анализировать для нескольких заданных значений N. Первый случай — это случай систем, имеющих фиксированное число областей основной памяти, которые вследствие предположения об очень большой рабочей нагрузке характеризуются очень маленькой вероятностью незанятости. Центры обслуживания замкнутой сети, моделирующей систему с постоянным значением N, представляют те ресурсы, которые доступны заданию, находящемуся в основной памяти. Этими ресурсами являются, обычно, центральный процессор (или процессоры) и каналы, управляющие устройствами внешней памяти. Другие центры, представляющие, например, периферийные устройства ввода-вывода, могут быть добавлены для получения более сложной сети. В этой сети затем выделяется подсеть, общее число заданий в которой поддерживается постоянным с помощью управляющей процедуры. Примером такой сети является модель системы Multics, исследованная Секино (1973).

Первым типом замкнутой сети, который будет рассмотрен в этом разделе, является модель с центральным обслуживающим прибором. Эта модель исследована Бузеном (1971), использовавшим подход к решению экспоненциальных замкнутых сетей, предложенный Гордоном и Ньюэллом (1967).

Пример 4.9. Моделируемая система — это ПРОП-система, рассмотренная в примере 4.8. Мы, однако, хотим исследовать ее производительность в предположении,

что степень мультипрограммирования постоянна. Таким образом, в нашей модели будут явно представлены только процессоры P_1, P_2 и P_3 на рис. 1.2, т. е. те, которые доступны заданию, загруженному в основную память.

Модель с центральным обслуживающим прибором с тремя центрами обслуживания показана на рис. 4.18. Каждый центр состоит из одного прибора и его очереди. Центральный прибор Σ_1 представляет центральный процессор, а остальные, Σ_2 и Σ_3,— соответственно процессоры барабана и диска. Возвратный путь вокруг центрального обслуживающего прибора имеет совершенно другой смысл, чем в прерываемой модели на рис. 4.14. Все дисциплины обслуживания в модели с центральным обслуживающим прибором непрерываемые. Возвратный путь совершают те задания, которые были обработаны вплоть до завершения и покидают систему. Для поддержания постоянного N каждое уходящее задание немедленно заменяется новым заданием, которое ставится в очередь центрального процессора Q_1. Так как все задания модели статистически одинаковы, эта управляющая процедура может быть реализована возвратным путем на рис. 4.18, по которому задание, покидающее систему, немедленно возвращается в нее. Наличие возвратного пути позволяет легко определить как пропускную способность, так и время ответа.

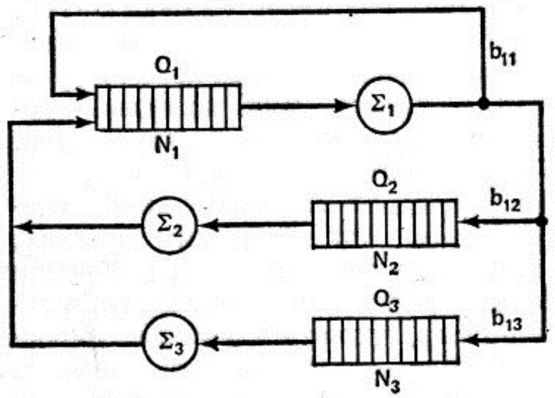


Рис. 4.18. Замкнутая сеть с центральным обслуживающим прибором для ПРОП-системы с рис. 1.2.

Как и в примере 4.8, мы предполагаем, что времена обслуживания во всех центрах распределены экспоненциально и что вероятности перехода в остальные центры по завершении обслуживания в некотором центре постоянны и одинаковы для всех заданий. Средние интенсивности обслуживания для всех центров тоже предполагаются постоянными. Эти предположения могут быть отчасти ослаблены без ухудшения разрешимости модели [см., например, Гордон и Ньюэлл (1967) и Баскетт и др. (1975)]. Для упрощения рассуждений мы выбрали более жесткие ограничения. В табл. 4.9 приведены параметры модели с рис. 4.18. Значение b_{11}, т. е. вероятность того, что задание пойдет по возвратному пути вокруг центрального прибора, может быть определено из b_{12} и b_{13}, так как $\sum_{i=1}^{3} b_{1i}=1$.

Как обычно, состояние сети с очередями с S центрами обслуживания определяется вектором $\mathbf{N}=(N_1, N_2, \ldots, N_s)$, где N_i — число заданий в i-м центре обслуживания. Найдем сначала решение для произвольной замкнутой сети с s экспоненциальными центрами обсуживания и N заданиями. Потом общие результаты будут применены к частному случаю модели с центральным обслуживающим прибором.

Так как мы имеем

$$\sum_{i=1}^{s} N_i = N \tag{4.89}$$

и N — константа, то число различных состояний равно числу распределений N заданий по S центрам, т. е.

$$\binom{N+s-1}{s-1}.$$

Если марковский процесс, характеризуемый этими состояниями, является неприводимым, т. е. если каждое состояние достижимо из любого другого состояния

с ненулевой вероятностью, то существует равновесное распределение вероятностей состояний (см. разд. 4.3.1). Гордон и Ньюэлл (1967) получили это распределение в предположении, что при равновесии частота переходов из состояния **N** в другие состояния равна частоте переходов из других состояний в состояние **N**.

Таблица 4.9

Параметры модели с центральным обслуживающим прибором на рис. 4.18

Название параметра	Обозначение	Значение
Среднее непрерываемое время ЦП	$1/\mu_1$	10 мс
Среднее время обслуживания барабана	$1/\mu_2$	25 мс
Среднее время обслуживания диска	$1/\mu_3$	100 мс
Вероятность запроса диска	b_{13}	0.1
Вероятность запроса барабана	b_{12}	0.8
Степень мультипрограммирования	N	3

Введем двоичную функцию

$$\begin{aligned} a(N_i) &= 0 \quad \text{для} \quad N_i = 0, \\ &= 1 \quad \text{для} \quad N_i > 0. \end{aligned} \tag{4.90}$$

Для каждого отдельного состояния **N** мы можем написать равновесное уравнение, которое называется *уравнением баланса* для состояния **N**:

$$\left[\sum_{k=1}^{s} a(N_k)\,\mu_k\right] p(\mathbf{N}) = \sum_{i=1}^{s}\sum_{k=1}^{s} a(N_k)\,a(N_i+1)\,\mu_i b_{ik} \times \\ \times p(N_1, \ldots, N_k-1, \ldots, N_i+1, \ldots, N_s). \tag{4.91}$$

Одно из уравнений (4.91) зависит от остальных. Тем не менее решение можно получить, если присоединить к уравнениям баланса (4.91) обычное условие:

$$\sum_{\text{все } \mathbf{N}} p(\mathbf{N}) = 1. \tag{4.92}$$

Правая часть (4.91) представляет частоту, с которой состояние **N** достигается из других состояний вследствие того, что завершение обслуживания задания в центре обслуживания i сопровождалось постановкой этого задания в очередь в центре обслуживания k. Присутствие члена $a(N_k)$ необходимо из-за того, что этот переход нельзя учесть, если в **N** мы имеем $N_k=0$.

Уравнения (4.91) и (4.92) могут быть решены с помощью разделения переменных. Пусть

$$p(\mathbf{N}) = C(N)\,x_1^{N_1} x_2^{N_2} \ldots x_s^{N_s}, \tag{4.93}$$

где $C(N)$ — коэффициент, значение которого зависит от N. Подставляя (4.93) в (4.91) и замечая, что $a(N_i+1)=1$ для всех i, имеем

$$\sum_{k=1}^{s}\left[a(N_k)\left(\mu_k - \sum_{i=1}^{s}\mu_i b_{ik}\frac{x_i}{x_k}\right)\right] = 0. \tag{4.94}$$

Поскольку (4.94) должно быть справедливо также и для тех состояний, в которых все задания находятся в одном и том же центре обслуживания, то все коэффициенты

при $a(N_k)$ при равновесии должны быть равны нулю. Таким образом,

$$\mu_k x_k = \sum_{i=1}^{s} \mu_i x_i b_{ik} \quad (k=1,\ 2,\ \ldots\ s). \tag{4.95}$$

Рассмотрим теперь модель с центральным обслуживающим прибором, имеющую S приборов и N заданий. В такой модели мы имеем $b_{ik}=0$ для $i\neq 1$ и $k\neq 1$. Следовательно, уравнения (4.95) при $k\neq 1$ будут типа

$$\mu_k x_k = \mu_1 x_1 b_{1k} \quad (k=2,\ \ldots\ s). \tag{4.96}$$

Если мы положим

$$y_k = \frac{\mu_1}{\mu_k} b_{1k} \quad (k=2,\ \ldots,\ s), \tag{4.97}$$

то можно переписать (4.96) следующим образом:

$$x_k = y_k x_1 \quad (k=2,\ \ldots,\ s), \tag{4.98}$$

и (4.93) примет вид

$$p(\mathbf{N}) = C(N)\, x_1^N y_2^{N_2} \ldots y_s^{N_s}. \tag{4.99}$$

Подставляя (4.99) в (4.92), получаем

$$C(N) = \left(x_1^N \sum_{\text{все } \mathbf{N}} \prod_{k=2}^{s} y_k^{N_k} \right)^{-1}. \tag{4.100}$$

Таким образом, если положить

$$G(N) = \sum_{\text{все } \mathbf{N}} \prod_{k=2}^{s} y_k^{N_k}, \tag{4.101}$$

то равновесные вероятности состояний (4.99) модели с центральным обслуживающим прибором будут задаваться так:

$$p(\mathbf{N}) = \frac{1}{G(N)} \prod_{k=2}^{s} y_k^{N_k}. \tag{4.102}$$

Когда эти вероятности известны, то можно вычислить наиболее интересные для нас индексы производительности. Средняя пропускная способность при равновесии равна величине b_{11}, умноженной на интенсивность выходного потока центрального прибора. Это выходная интенсивность совпадает с μ_1, когда центральный прибор занят ($N_1>0$), и равна нулю, когда он простаивает. Таким образом, если вспомнить определение коэффициента использования центрального процессора ρ_1, мы имеем

$$\overline{T} = b_{11}\mu_1 \sum_{\substack{\text{все } \mathbf{N} \\ \text{при } N_1>0}} p(\mathbf{N}) = b_{11}\mu_1\rho_1. \tag{4.103}$$

Если мы разорвем возвратный путь вокруг центрального прибора, но сохраним управление степенью мультипрограммирования, как в замкнутой сети, то значение $\overline{T}$, задаваемое (4.103), будет равно значению средней интенсивности появления заданий. Используя результат Литтла (4.41), получаем выражение для среднего времени ответа или времени прохождения:

$$\overline{t} = \frac{N}{\overline{T}}. \tag{4.104}$$

Используемое в (4.103) выражение ρ_i может быть преобразовано в выражение, которое, как мы увидим ниже, вычисляется более просто. Заметим что показатели N_k, появляющиеся в (4.101), представляют собой все возможные значения числа заданий в центре обслуживания k. Это число колеблется от нуля (для тех состояний, в которых центр k пуст) до N (в состоянии, когда все центры, кроме k, пусты). Если рассматриваются только состояния, в которых $N_1>0$, то последний случай ($N_k=N$) никогда

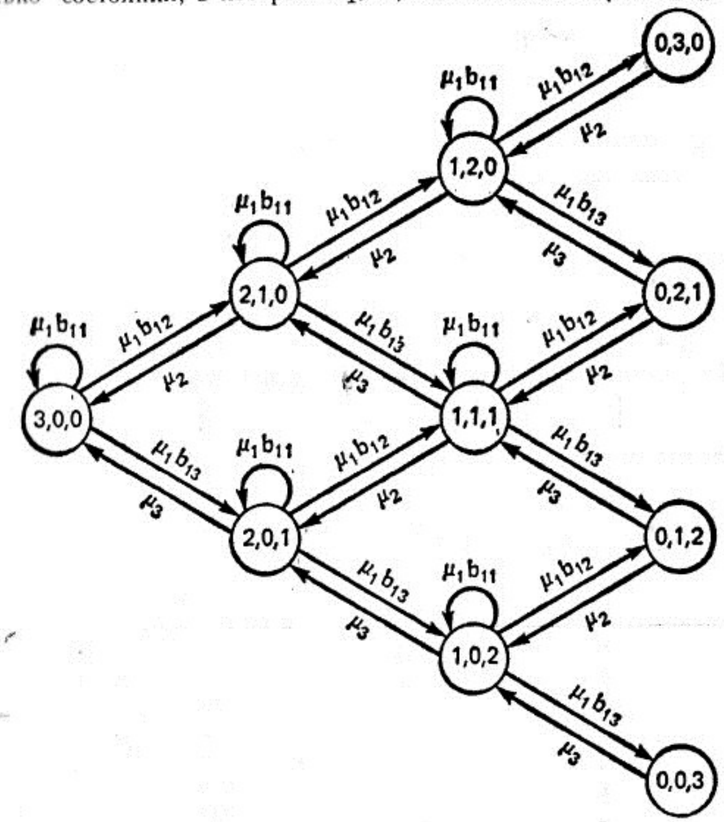


Рис. 4.19. Диаграмма интенсивностей переходов между состояниями для модели с центральным обслуживающим прибором на рис. 4.18 при $N=3$. Каждый узел содержит значения N_1, N_2, N_3, соответствующие представляемому им состоянию.

не встретится; так как минимальное значение $N_1=1$, то N_k не может быть больше $N-1$. Таким образом, мы имеем

$$\sum_{\substack{\text{все } \mathbf{N} \\ \text{при } N_1>0}} \prod_{k=2}^{s} y_k^{N_k} = G(N-1), \tag{4.105}$$

из (4.103) и (4.102)

$$\rho_1 = \frac{G(N-1)}{G(N)}. \tag{4.106}$$

Коэффициент использования центра обслуживания i ($i=2,\ \ldots,\ s$) будет

$$\rho_i = \sum_{\substack{\text{все } N \\ \text{при } N_i>0}} p(\mathbf{N}) = \frac{y_i G(N-1)}{G(N)} = y_i\rho_1. \tag{4.107}$$

Отметим, что (4.107) можно получить, приравняв среднюю интенсивность поступления заданий для центра обслуживания i к средней интенсивности выходного потока для этого же центра и вспомнив определение (4.97):

$$b_{1i}\mu_1\rho_1 = \mu_i\rho_i. \tag{4.108}$$

Используем эти результаты для нашей модели с тремя обслуживающими приборами и с $N=3$. В этом случае модель имеет 10 состояний. Ее диаграмма интенсивностей переходов между состояниями показана на рис. 4.19. Диаграмма построена исходя

Таблица 4.10

Алгоритм равновесного решения модели с центральным прибором при $s=3$

(a)

N	Y_1	Y_2	Y_3
0	1	1	1
1	1	$1+y_2$	$1+y_2+y_3$
2	1	$1+y_2(1+y_2)$	$1+y_2(1+y_2)+y_3(1+y_2+y_3)$
3	1	$1+y_2[1+y_2(1+y_2)]$	$1+y_2[1+y_2(1+y_2)]+y_3[1+y_2(1+y_2)+y_3(1+y_2+y_3)]$
⋮	⋮	⋮	⋮

(b)

$y_1=\frac{\rho_1}{\rho_1}=1$ $\quad y_2=\frac{\rho_2}{\rho_1}=2$ $\quad y_3=\frac{\rho_3}{\rho_1}=1$

N	Y_1	Y_2	$Y_3=G$	ρ_1
0	1	1	1	0
1	1	3	4	0.25
2	1	7	11	0.363
3	1	15	26	0.423
4	1	31	57	0.456
5	1	63	120	0.475
6	1	127	247	0.496
⋮	⋮	⋮	⋮	⋮

из того, что интенсивность поступления заданий из центра 1 в центр 2 равна $\mu_1 b_{12}$, а интенсивность переходов из 2 в 1—μ_2. Аналогично, переходы из 1 в 3 и из 3 в 1 происходят соответственно с интенсивностями $\mu_1 b_{13}$ и μ_3.

Уравнение баланса для состояния (3,0,0) можно написать на основании диаграммы на рис. 4.19 следующим образом:

$$(\mu_1 b_{11}+\mu_1 b_{12}+\mu_1 b_{13})\,p\,(3,0,0)=\mu_1 b_{11}p\,(3,0,0)+\mu_2 p\,(2,1,0)+\mu_3 p\,(2,0,1). \tag{4.109}$$

Заметим, что левая часть (4.109) в точности равна $\mu_1 p\,(3,0,0)$; мы могли бы написать это прямо, используя (4.19).

Для вычисления ρ_1, значение которого требуется для получения коэффициентов использования других центров обслуживания, а также значений $\bar{T}$ и $\bar{t}$, мы должны по (4.106) вычислить $G(N)$ и $G(N-1)$. Из (4.101) имеем

$$\begin{aligned} G(2)&=1+y_2+y_3+y_2^2+y_2y_3+y_3^2,\\ G(3)&=G(2)+y_2^3+y_2^2y_3+y_2y_3^2+y_3^3. \end{aligned} \tag{4.110}$$

Заметим, что в $G(2)$ из (4.110) выписаны только те состояния с рис. 4.19, для которых $N_1>0$. Состояние (3, 0, 0), например, дает член $y_2^0\,y_3^0=1$ и для $G(2)$, и для $G(3)$.

При значениях параметров, приведенных в табл. 4.9, получаем [см. (4.37)] $y_2=2$ и $y_3=1$. Таким образом, из (4.110) мы имеем $G(2)=11$ и $G(3)=26$. Уравнение (4.106) дает $\rho_1=0.423$. Из (4.107) получаем $\rho_2=0.846$ и $\rho_3=0.423$. Наконец, средняя пропускная способность, задаваемая (4.103), будет $\bar{T}=4.23$ задание/с, а среднее время ответа, задаваемое (4.104), будет $\bar{t}=0.709$ с.

Эффективный вычислительный алгоритм для расчета $G(N)$ был предложен Бузеном (1973). Рекурсивно определяем две функции $Y_2(N)$ и $Y_3(N)$ следующим образом:

$$\begin{aligned} &Y_2(0)=1,\\ &Y_3(0)=1,\\ &Y_2(N)=1+y_2Y_2(N-1),\\ &Y_3(N)=Y_2(N)+y_3Y_3(N-1) \quad (N=1,2,\ldots). \end{aligned} \tag{4.111}$$

Читатель может легко убедиться, что для всех N верно $Y_3(N)=G(N)$. Вычисление, таким образом, может быть организовано, как в табл. 4.10 (a) . Каждая запись в столбце Y_2 и Y_3 получается умножением находящейся непосредственно сверху записи соответственно на y_2 или y_3 и сложением с записью, расположенной непосредственно слева в той же строке. Например, $Y_3(3)=Y_2(3)+y_3Y_3(2)$, как указывает (4.111). В табл. 4.10 (b) показаны результаты выполнения этого алгоритма для значений параметров из табл. 4.9.

Этот алгоритм может использоваться для решения моделей с центральным обслуживающим прибором и произвольным числом приборов. Рекурсивные определения (4.111) могут быть обобщены следующим образом:

$$\begin{aligned} &Y_i(0)=1,\\ &Y_1(N)=1,\\ &Y_i(N)=Y_{i-1}(N)+y_iY_i(N-1) \quad (N=1,2,\ldots;\ i=2,\ldots,s), \end{aligned} \tag{4.112}$$

и всегда

$$G(N)=Y_s(N). \tag{4.113}$$

Заметим, что нумерация центров обслуживания не влияет на результат, единственное исключение — центральный прибор, которому должен быть присвоен номер 1. Из (4.101) легко видеть, что функция $G(N)$ симметрична относительно y_2 и y_3. ■

Модель средних значений, исследованную в разд. 4.2.3, можно рассматривать как частный случай модели с центральным прибором и двумя центрами обслуживания. Уравнение (4.8) эквивалентно (4.108), поскольку средними интенсивностями выходных потоков процессорного и барабанного центров обслуживания будут соответственно $\mu_1\rho_1$ и $\mu_2\rho_2$, а вероятность перехода «центральный процессор — барабан» будет $b_{12}=n_{бр}/(1+n_{бр})$. Аналогично, (4.9) эквивалентно (4.103), так как в модели с двумя обслуживающими центрами $b_{11}=1/(1+n_{бр})$. Метод, описанный в примере 4.2 для решения модели средних значений, предполагает, что скорости работы центрального процессора и барабана являются известными функциями от среднего числа заданий в соответствующих центрах обслуживания. Исследование, проведенное в примере 4.9, дает нам способ вычисления $\rho_1(N)$ и $\rho_2(N)$ для такой системы и таких параметров рабочей нагрузки, как параметр μ_1 [который определяется как $(n_{бр}+1)/t_{цп}$; см. (4.10)], μ_2 [определяющийся как $1/R_{бр}$; см. (4.12)] и b_{12} [который будет равен $n_{бр}/(1+n_{бр})$].

Применив этот анализ к модели с центральным прибором и с двумя центрами обслуживания и со значением параметров из табл. 4.5, получим $\rho_1=0.878$, $\rho_2=0.702$ и $\bar{T}=3.512$ заданий/с. Различия между этими результатами и результатами примера 4.2 должны быть отнесены за счет того, что в этом примере используется кривая для скорости барабана $\bar{f}_2(\bar{N}_2)$, отличающаяся от той, что предлагается в модели с центральным прибором, а также за счет того, что времена обслуживания в обоих приборах предполагаются постоянными.

Другим важным приложением замкнутых сетей является оценка производительности диалоговых систем аналитическими методами. В диалоговой системе с N активными терминалами имеется в точности N заданий. Во всякий момент времени N_s из этих заданий находятся в «системе», а соответствующие N_s пользователей ожидают ответа на своих терминалах. Оставшиеся $N_t=N-N_s$ заданий находятся «на терминалах», и система не может возобновить свою работу на любом из них до тех пор, пока она не получит входного сообщения с соответствующего терминала (см. рис. 1.4). Таким образом, коллектив пользователей может быть представлен обслуживающим прибором с N параллельными и независимыми каналами без очередей. Каждый канал моделирует отдельного пользователя.

Рассмотрим теперь простую модель замкнутой сети для диалоговой установки, исследованную Клейнроком (1968).

Пример 4.10. Простейшей моделью в виде замкнутой сети для диалоговой установки является модель, представляющая всю систему в виде единственного центра обслуживания. Если эта система с разделением времени, то дисциплина обслуживания в центре должна быть квантованной. Выберем для упрощения решения модели дисциплину с разделением времени процессора (см. разд. 4.3.3.) и предположим, что все задания статистически одинаковы. Модель изображена на рис. 4.20, а ее параметры приведены в табл. 4.11.

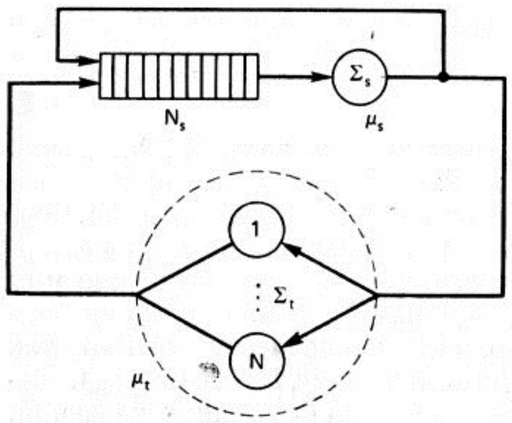


Рис. 4.20. Замкнутая сеть для диалоговой установки.

Большинство замечаний, сделанных в примере 4.7, применимы также к использованию этой модели для оценки производительности диалоговой системы. В частности, в модели предполагается, что система является мультипрограммной, что можно пренебречь временами попеременной загрузки, что квант времени очень мал по сравнению со средним временем обслуживания.

Если и времена обслуживания и терминальные времена распределены экспоненциально, то стохастический процесс $N_s(t)$ является конечной марковской цепью с непрерывным временем. Цепь конечна, поскольку $0\leqslant N_s(t)\leqslant N$. Диаграмма интенсивностей переходов между состояниями, приведенная на рис. 4.21, показывает, что данная марковская цепь есть в действительности процесс рождения — гибели (см. разд. 4.3.2).

Таблица 4.11

Параметры модели — замкнутой сети для диалоговой системы с рис. 4.20

Название параметра	Обозначение	Значение
Среднее системное время обслуживания (среднее время однопрограммного выполнения)	$1/\mu_s$	0.5с
Среднее терминальное время (или время „обдумывания“)	$1/\mu_t=\bar{t}_t$	10с
Число активных терминалов (пользователей)	N	10

Средние интенсивности на рис. 4.21 могут быть выведены следующим образом. Пусть в момент времени t модель находится в состоянии N_s. Каждое из N_s заданий в обслуживающем приборе Σ_s обрабатывается со скоростью, равной $1/N_s$ от скорости прибора. Вероятность того, что обслуживание какого-либо задания завершится в течение очень короткого интервала $(t, t+h)$, равна $N_s(\mu_s/N_s)h$, а соответствующая средняя интенсивность завершения будет μ_s. Так как каждый канал в Σ_t имеет среднюю интенсивность обслуживания μ_t и так как $N-N_s$ каналов в состоянии N_s занято, то интенсивность появления в обслуживающем центре системы будет $(N-N_s)\,\mu_t$.

Распределение терминальных времен (или времен «обдумывания»), измеренное Шерром (1967) для установки CTSS в Массачусетском технологическом институте,

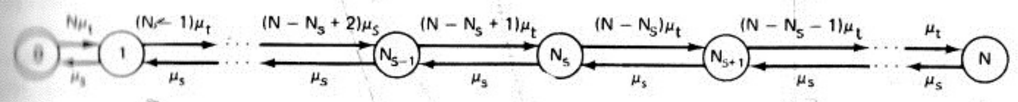


Рис. 4.21. Диаграмма интенсивностей переходов между состояниями для модели — замкнутой сети на рис. 4.20.

похоже на экспоненциальное для времен больше 10 с, но существенно отличается от экспоненциального для более коротких времен. Их среднее равно 35.2 с, медиана — 11 с, а коэффициент вариации — 0.65. Брайен (1967) и Парупуди и Виноград (1972) нашли, что среднее время обдумывания находится между 10 и 25 с.

Уравнение баланса можно вывести непосредственно из диаграммы на рис. 4.21:

$$\begin{aligned} &p_0 N\mu_t = p_1\mu_s,\\ &p_{N_s}[(N-N_s)\,\mu_t+\mu_s] = p_{N_s-1}\,(N-N_s+1)\,\mu_t + p_{N_s+1}\mu_s \quad (N_s=1,\ldots N-1),\\ &p_{N-1}\mu_t = p_N\mu_s. \end{aligned} \tag{4.114}$$

Читатель может легко убедиться, что (4.114) можно записать в следующем более простом и компактном виде:

$$p_{N_s}(N-N_s)\,\mu_t = p_{N_s+1}\mu_s \quad (N_s=0,\ 1,\ \ldots\ N-1). \tag{4.115}$$

Одно из N уравнений (4.115) зависит от остальных $N-1$. С помощью соответствующих подстановок мы можем выразить все вероятности состояний через P_0:

$$\begin{aligned} p_{N_s} &= p_{N_s-1}\,(N-N_s+1)\frac{\mu_t}{\mu_s} = p_{N_s-2}\,(N-N_s+1)\quad (N-N_s+2)\left(\frac{\mu_t}{\mu_s}\right)^2 = \\ &= \ldots = p_0\frac{N!}{(N-N_s)!}\left(\frac{\mu_t}{\mu_s}\right)^{N_s} \quad (N_s=0,\ 1,\ldots N), \end{aligned} \tag{4.116}$$

Равновесная вероятность того, что система пуста, p_0, получается наложением условия

$$\sum_{N_S=0}^{N} p_{N_S} = p_0 \sum_{N_S=0}^{N} \frac{N!}{(N-N_S)!} \left(\frac{\mu_t}{\mu_s}\right)^{N_S} = 1. \qquad (4.117)$$

Из (4.117) получаем коэффициент использования системы $\rho_s = 1 - p_0$. Средняя пропускная способность модели системы совпадает с интенсивностью выходного потока Σ_s:

$$\bar{T} = \mu_s \rho_s. \qquad (4.118)$$

Каждый из пользователей, чьи задания находятся на терминалах, генерирует μ_t запросов в единицу времени. Вероятность того, что задание находится на терминале, задается формулой $(1/\mu_t)$ $(\bar{t}+1/\mu_t)$, где $\bar{t}$ — среднее время в Σ_s, а $\bar{t}+1/\mu_t$ — среднее время взаимодействия. Таким образом, средняя интенсивность появления заданий в Σ_s будет

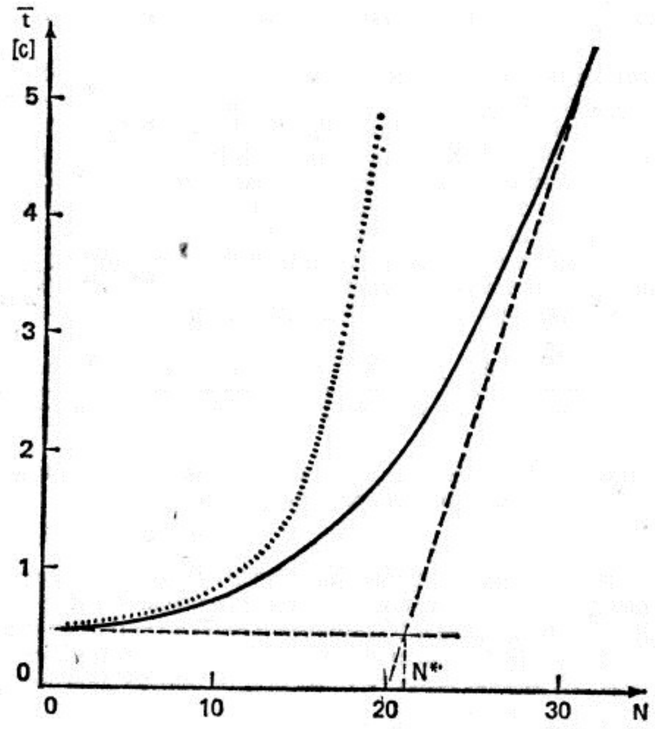


Рис. 4.22. Среднее время ответа системы для модели на рис. 4.20 и таблице 4.11 относительно числа активных терминалов.

$$\lambda = N\mu_t \frac{1/\mu_t}{\bar{t}+1/\mu_t} = \frac{N}{\bar{t}+\bar{t}_t}, \qquad (4.119)$$

где $\bar{t}_t = 1/\mu_t$.

При равновесии λ должно быть равно $\bar{T}$. Из (4.118) и (4.119) получаем

$$\bar{t} = \frac{N}{\mu_s \rho_s} - \bar{t}_t. \qquad (4.120)$$

Для значений параметров из табл. 4.11 имеем $p_0 = 0.538$, $\rho_s = 0.462$, $\bar{T} = 0.924$ задание/с и $\bar{t} = 0.822$ с.

Один из наиболее частых вопросов, задаваемых относительно диалоговой системы, состоит в том, какое максимальное число терминалов эта система может обеспечивать. Этот вопрос в том виде, как он сформулирован в предыдущем предложении, бессмыслен по двум причинам: во-первых, должно быть дано точное определение того, что подразумевается под *обеспечением*; во-вторых, должна быть четко указана рабочая нагрузка, генерируемая терминалами. В контексте нашей модели мы будем говорить, что диалоговая система в состоянии обеспечить N терминалов, которые генерируют статистически одинаковые задания, характеризующиеся экспоненциальными определениями системных и терминальных времен с параметрами μ_s и μ_t, если среднее время ответа $\bar{t}$ меньше заданного максимального значения $\bar{t}_{\max}$. Заметим, что на самом деле и μ_s и в какой-то мере μ_t тоже зависят от параметров системы. Однако поскольку эти параметры системы предполагаются постоянными, то можно считать, что μ_s и μ_t являются параметрами только рабочей нагрузки.

Если мы графически изобразим функцию $\bar{t}(N)$ для нашей модели со значениями из табл. 4.11, то получим кривую, показанную на рис. 4.22. Начиная с некоторого числа терминалов, среднее время ответа системы возрастает значительно быстрее, чем при меньших N, асимптотически стремясь к прямой с положительным наклоном. Следуя Клейнроку (1968), мы определим точку $(N^*, \bar{t}^*)$, в которой асимптота пересекается с горизонтальной прямой $\bar{t} = \bar{t}(1)$, как системную *точку насыщения*.

В хорошо спроектированной системе скорость обработки (которой пропорциональна величина μ_s) должна быть не настолько высокой, чтобы $\bar{t}^*$ оказалось существенно меньше $\bar{t}_{\max}$. Таким образом, определенное выше значение N^* является разумной оценкой максимального числа терминалов, работу которых может обеспечить система.

Значением $\bar{t}^* = \bar{t}(1)$ будет, конечно, $1/\mu_s$. Уравнение асимптоты для кривой $\bar{t}(N)$ можно получить, если заметить, что при $N \to \infty$ мы имеем $p_0 \to 0$ и $\rho_s \to 1$. Таким образом, (4.120) дает для $N \gg N^*$

$$\bar{t} \cong \frac{N}{\mu_s} - \bar{t}_t, \qquad (4.121)$$

а из определения N^* имеем

$$N^* = 1 + \frac{\mu_s}{\mu_t}. \qquad (4.122)$$

Для нашей модели $N^* = 21$ терминалу.

Для $N \ll N^*$ кривую $\bar{t}(N)$ можно аппроксимировать функцией

$$t \cong \frac{1}{\mu_s} \cdot \frac{1}{1-(N-1)(\mu_t/\mu_s)}, \qquad (4.123)$$

которая получается из (4.117), (4.120) и из условия $(N-1)\mu_t/\mu_s \ll 1$, выводящегося из (4.122). Функция (4.123) на рис. 4.22 изображена пунктирной кривой.

Рассмотренная в примере 4.10 модель может также использоваться для оценки мультипрограммных систем с постоянной степенью мультипрограммирования N и с числом каналов ввода-вывода (или независимых путей к устройствам ввода-вывода), значительно большим чем N. Если последнее предположение выполнено, то вероятность того, что задание потребует прибор ввода-вывода, обслуживающий другое задание, очень мала. Таким образом, очереди в приборах ввода-вывода можно игнорировать, и систему можно моделировать так же, как и

на рис. 4.20. Однако теперь Σ_t представляет N параллельных одинаковых приборов ввода-вывода, а Σ_s — центральный процессор. Сеть больше не является моделью установки, а представляет некую систему. Если эта система с разделением времени, то можно выбрать в качестве дисциплины для центра обслуживания центрального процессора разделение времени процессора (дисциплину PS), как это было сделано в примере 4.10 для центра обслуживания системы.

Коллектив пользователей может быть явно промоделирован добавлением или внешнего источника (открытая сеть) или еще одного цикла, содержащего центр обслуживания, такой, как на рис. 4.20 (замкнутая сеть). В обоих случаях, если постулировано наличие управляющей процедуры, поддерживающей степень мультипрограммирования постоянной, решение модели не зависит от характеристик источников или дополнительного центра обслуживания. Однако добавление в модель коллектива пользователей позволяет нам формально определить и вычислить такие индексы, как $\bar{t}$ и $\bar{T}$. Заметим, что степень мультипрограммирования можно поддерживать постоянной, только если в модель включена очередь заданий, ожидающих памяти и если коллектив пользователей настолько активен, что эта очередь никогда не оказывается пустой, когда какое-то задание покидает модель системы.

Бойз и Уорн (1975) использовали эту модель для оценки диалоговой мультипрограммной системы, снабженной виртуальной памятью и обеспечивающей работу N_u графических терминалов. В этой системе (при $N_{max}=3$) имеется 11 путей к устройствам ввода-вывода и большинство потоков ввода-вывода определяется страничным обменом. Таким образом, предположение о том, что имеются одинаковые обслуживающие приборы ввода-вывода без очередей, не является слишком нереалистичным. Наложенное на модель ограничение, состоящее в том, что запросы на ввод-вывод от некоторого задания не могут перекрываться с вычислениями, необходимыми для этого же задания, имеет смысл для выборки страницы, но не для запоминания страницы (поэтому запоминание страницы игнорируется).

При экспоненциальных временах обслуживания для обоих приборов коэффициент использования центрального процессора может быть вычислен из (4.117). Если b — вероятность того, что задание, покидающее обслуживающий центр центрального процессора, покинет и систему, то средняя пропускная способность будет $T=b\mu_s\rho_s$, а среднее время ответа $\bar{t}=(N_u/\bar{T})-\bar{t}_u$. Отметим, что теперь μ_s и ρ_s — средняя скорость обслуживания и коэффициент использования центрального процессора; N_u — число терминалов, а $\bar{t}_u$ — среднее время, которое задание проводит на терминале.

Аналогичная модель была использована Секино (1973) для мультипроцессорной системы с виртуальной памятью Multics. Модель Секино представляет собой замкнутую сеть из двух циклов. Внутренний цикл состоит из двух центров обслуживания. Центр обслуживания центрального процессора содержит очередь для готовых заданий и многоканальный обслуживающий прибор. Каждый канал моделирует центральный процессор. Другой центр состоит из очереди и единственного обслуживающего центра, представляющего страничное устройство. Число заданий во внутреннем цикле предполагается постоянным. Внешний цикл содержит центр обслуживания с прибором, таким же, как Σ_t на рис. 4.20, который представляет терминалы пользователей. За этим центром следует очередь, в которой появляющиеся в системе задания ожидают до тех пор, пока они не будут загружены в основную память.

4.4. КАЛИБРОВКА И КОНТРОЛЬ АНАЛИТИЧЕСКОЙ МОДЕЛИ

Проведенное в разделе 3.3.1 рассмотрение точности модели, ее калибровки и контроля приложимо не только к имитационным, но и к аналитическим моделям. В частности, точность аналитической модели можно определить согласно рис. 3.16. Перечисленные в разд. 3.3.1 три типа ошибок (если предположить, что значения системных индексов производительности $\mathcal{P}$ не испорчены более существенными ошибками) являются основными погрешностями, которые могут вызывать расхождение между полученными из аналитической модели значениями $\mathcal{P}'$ и значениями $\mathcal{P}$.

К ошибкам формулировки обычно относят чересчур низкую степень детализации или слишком грубые допущения, которые приходится делать для того, чтобы модель была разрешимой.

Ошибки решения при символьных решениях могут быть вызваны ошибками в математических преобразованиях или использованными методами аппроксимации. Примерами неточностей последнего типа являются погрешности, возникающие вследствие применения диффузионной аппроксимации для решения моделей с очередями. При численном решении аналитической модели ошибки решения обычно следует относить за счет использованных численных методов. Например, точность решения для итеративных методов обычно зависит от числа произведенных итераций.

Наконец, на *неточности параметров* могут влиять оценки значений параметров модели и распределений, т. е. значения $\mathcal{P}'$. Даже если моделируемая система доступна и все модельные параметры можно измерить, все равно в некоторых случаях может оказаться полезным определить некоторые из них, попытавшись подогнать вычисленную производительность к измеренной. Например, таким образом могут быть определены значения для моделей на рис. 4.16 и 4.18.

Эмпирическая оценка параметров рабочей нагрузки и их распределений должна производиться с использованием аналитических методов, описанных в разд. 2.4.3, 2.6 и 2.7, а также методов, применявшихся в работах Фукса и Джексона (1970), Андерсона и Сарджента (1972) и Льюиса и Ю (1972). Статистические тесты согласия эмпирических и аналитических распределений описаны в различных книгах по статистическим методам [например, Снедекор и Кокрен (1967)].

Эти тесты не обязательны для калибровки детерминированных моделей, которые не основываются на предположениях о распределениях. Однако те же методы эмпирической оценки численных значений параметров могут использоваться для детерминированных моделей и для моделей с очередями.

Калибровка и контроль в основном направлены против ошибок формулировки. С помощью калибровки и контроля можно попытаться определить эти ошибки и устранить их причины. Однако все три типа ошибок могут быть учтены в процедуре, изображенной на рис. 3.16. Перед тем как применить процедуру калибровки, исследователь должен попытаться отделить ошибки решения и ошибки параметров от ошибок, обусловленных формулировкой модели.

Ошибки решения часто удается обнаружить, если есть возможность применить альтернативный метод решения модели. Для аналитических моделей один такой метод существует всегда: можно имитировать модель (но не систему) и результаты сравнить с аналитическим решением. И наоборот, если модель имитационная, то бывает возможно найти не очень отличную от нее разрешимую аналитическую модель и использовать результаты этой аналитической модели для проверки точности имитационных экспериментов. Простой пример такой процедуры приведен на рис. 3.18.

При калибровке аналитической модели необходимо измерение системы или, если оцениваемая система недоступна, имитация системы (но не модели). Если используется имитация, то имитационная модель должна быть пригодна по крайней мере при условиях, соответствующих значениям параметров, выбранным для процедуры калибровки. Читателю следует постоянно помнить, что имитационное моделирование, не проконтролированное измерениями, не является достаточной гарантией пригодности аналитической модели.

В общем случае для моделей с очередями не существует явных калибровочных параметров. Среди немногих исключений мы можем указать на поправку, введенную Шерром (1967) в его модель для того, чтобы учесть эффект попеременной загрузки. В примерах детерминированных моделей, описанных в разд. 4.2, калибровочные параметры явно не заданы, но некоторые из параметров модели могут быть легко интерпретированы и использованы как параметры калибровки.

Контроль определен как набор операций, предназначенных для того, чтобы сделать область пригодности модели настолько широкой, насколько это нужно для применений модели. В настоящее время не известно никаких процедур контроля для аналитических моделей. Как и для имитационных моделей, область пригодности аналитической модели может до некоторой степени зависеть от точного выбора условий калибровки. В истории моделей вычислительных систем с очередями пока только несколько моделей были проверены экспериментально. Одним из наиболее исчерпывающих сравнений аналитических результатов с результатами измерений и имитации была работа Шерра (1967). Его модель системы CTSS с разделением времени представляла собой

циклическую систему с очередями, как на рис. 4.20. Кроме нереалистичных допущений о распределениях, обусловленных требованиями разрешимости в символьном виде, недостатки этой модели состоят в том, что в ней не отражены попеременная загрузка (за исключением постоянного времени на попеременную загрузку и вспомогательные действия, добавленного к временам обслуживания), конечные и различные кванты (предполагается дисциплина разделения времени процессора) и стратегия планирования (стратегия FB_9 с множественными входами в очереди, основывающаяся на запросах объема памяти, с переменными во времени приоритетами и сложной процедурой предпочтения). Тем не менее полученные Шерром аналитические результаты определенно ближе к данным измерений и имитации, чем можно было ожидать. В гл. 4 [Шерр (1967)] графически сравниваются значения следующих индексов, полученные аналитическим и имитационным методами, а также измерением: отношение среднего времени ответа к среднему времени ЦП на одно взаимодействие как функция среднего числа пользователей, ведущих диалог; функция ответа системы; функция плотности вероятностей времени ответа; коэффициент использования центрального процессора как функция среднего числа пользователей, ведущих диалог; коэффициенты использования диска и барабана для цепей попеременной загрузки как функции среднего числа пользователей, ведущих диалог.

Результаты Шерра, так же как и результаты, опубликованные несколькими другими авторами [см. Мунц (1975)], показывают, видимо, что даже очень грубые модели диалоговых систем могут обеспечить разумную точность и относительно большую область пригодности, по крайней мере если считать приемлемыми ошибки, содержащиеся в этих результатах. У Хьюза и Мо (1973) описана аналитическая модель медиалоговой системы, из которой следуют аналогичные выводы (см. примеры 7.9 и 7.13).

Одним из наиболее спорных аспектов формулировки вероятностной модели является выбор распределений параметров модели. Несмотря на существенный прогресс в области моделей с очередями, этот выбор для большинства моделей ограничен несколькими типами распределений (если требуется решение в символическом виде). Какое влияние оказывает этот выбор на значения индексов производительности, выдаваемых данной моделью? Каков, например, эффект замены всех действительных распределений времен обслуживания в сетях с очередями на экспоненциальные?

Частный ответ на этот вопрос для специального случая непрерываемой модели с единственным центром обслуживания можно найти на рис. 4.10 и 4.11, на которых видно влияние коэффициента вариации времен обслуживания на среднее время ответа. Читатель может удостовериться, что $\bar{t}$ существенно возрастает при увеличении изменчивости времен обслуживания, которая представляется коэффициентом вариации c.

Этот результат подтверждается проведенными Бойзом и Уорном (1975) аналитическими и имитационными исследованиями замкнутой сети с рис. 4.20. График коэффициента использования ρ_s обслуживающего прибора Σ_s (центрального процессора) как функции от коэффициента вариации времен обслуживания центрального процессора изображен на рис. 4.23. Заметим, что точки, соответствующие распределению Эрланг-2 и гиперэкспоненциальному, получены соответственно

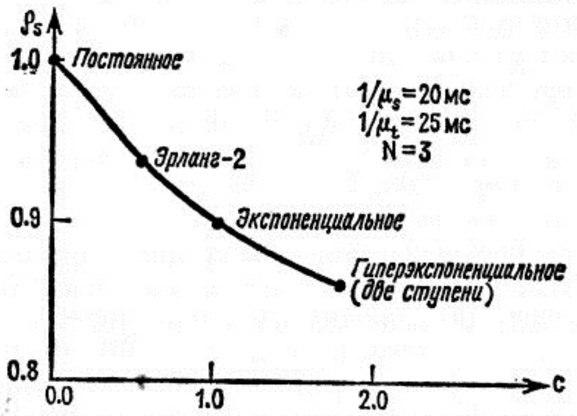


Рис. 4.23. Зависимость коэффициента использования ρ_s от коэффициента вариации c времен обслуживания центра обслуживания Σ_s для модели на рис. 4.20. [Источник: Бойз и Уорн (1975), с разрешения ACM, INC.]

только из одного из многих распределений с одинаковыми коэффициентами вариации. Максимальная разница между значениями ρ_s, изображенными на рис. 4.23, не превышает 15% для значений c в диапазоне (0, 1.8).

Таким образом, распределения времен обслуживания влияют на значения индексов производительности, предсказываемые моделью. Численная величина этого влияния зависит от типа модели и от значений параметров. Замена действительных распределений распределениями, имеющими меньший коэффициент вариации, приводит к оптимистическим результатам. Результаты оказываются пессимистическими, если принятые в модели распределения имеют коэффициенты вариации c больше, чем действительные.

Исследования чувствительности вроде только что упомянутых исключительно важны для оценщика производительности, да и вообще для тех, кто интересуется применением аналитических моделей. К сожалению, пока только немногие модели были проанализированы с этой точки зрения.

Задачи

4.1. Сформулируйте детерминированную модель мультипрограммной ПРОП-системы, исследованной в примере 4.9. Используя данные из табл. 4.9, вычислите значения среднего времени ответа и средней пропускной способности вашей модели и сравните их со значениями, полученными в примере 4.9 из модели с очередями на рис. 4.18.

4.2. В однопрограммной системе с разделением времени процесс полностью загружается в основную память до начала его выполнения и разгружается из памяти, когда он заканчивается или когда истекает его квант времени. Центральный процессор остается свободным, когда процесс выдает запрос на диск. Попеременная загрузка может быть совмещена с вычислением и происходит между основной памятью и барабаном. Постройте детерминированную модель системы для вычисления среднего времени ответа. Предположите, что средний процесс имеет характеристики из табл. 4.9 и что его среднее время попеременной загрузки 10 мс. Определите все допущения, которые вы сделали, и обсудите их важность и близость к действительности.

4.3. Центральный процессор может совершать одно обращение к памяти каждые 100 нс. Основная память состоит из четырех модулей, соединенных четырьмя путями так, что обращения к последовательным адресам на самом деле производятся обращением к последовательным модулям. Модули представляют собой полупроводниковые блоки памяти со временем чтения 400 нс и временем записи, равным времени чтения. Каждый модуль имеет независимый доступ, и цикл модуля сдвинут относительно цикла предыдущего модуля на 100 нс. Таким образом, максимальная ширина полосы пропускания (скорость передачи) памяти равна 10^7 слов/с, т. е. 1 слово каждые 100 нс. Если положить, что центральный процессор выдает обращения с максимальной частотой и что предыдущее обращение было к модулю i (i=0, 1, 2, 3), то вероятность того, что следующий модуль будет $i+1 \pmod 4$, есть $p_{i,(i+1) \bmod 4}=0.55$. Причем вероятности обращений к остальным модулям все равны друг другу. Постройте марковскую модель системы ЦП — память и определите действительную ширину полосы пропускания памяти.

4.4. Исследуйте непрерываемую модель с одним обслуживанием прибором для аналоговой системы с пуассоновским входным потоком (λ=2 взаимодействия/с) и гиперэкспоненциальным распределением из двух ступеней для времен обслуживания. Параметры этого распределения [см. (4.49)] равны: $\omega_1=0.92$, $\omega_2=0.08$, $\mu_1=0.92\ c^{-1}$ и $\mu_2=0.051\ c^{-1}$. Выпишите равновесные уравнения для этой модели и решите их. Выведите из (4.49) выражение для $\overline{t_s}$, задаваемое формулой (4.50). Покажите, что значение $\overline{t}$ равно вычисленному по результатам анализа $MG/1$-системы.

4.5. Среднее время ожидания для задания с временем обслуживания t_s в $MG/1$-модели с очередями с дисциплиной SPTF будет

$$\overline{t_w(t_s)}=\frac{\lambda}{\mu^2\left[1-\lambda\int_0^{t_s} x\,dF_{t_s}(x)\right]^2},$$

где $F_{ts}(x)$ — функция распределения времен обслуживания. Начертите график функции ответа этой системы в предположении, что $F_{ts}(x)$ — экспоненциальная функция распределения со средним $1/\mu$, используйте при этом данные из табл. 4.7. Сравните функцию ответа при дисциплине SPTF с такой же функцией в $M/M/1$-системе с дисциплиной распределения FCFS и такими же значениями параметров. Какую из этих двух дисциплин вы предпочли бы для диалоговой системы? Каков главный недостаток данной дисциплины? Насколько, по вашему мнению, этот недостаток серьезен? Можете ли вы предложить непрерываемую дисциплину обслуживания, которая, по возможности сохраняя преимущества дисциплин FCPS и SPTF, ослабит соответствующие недостатки этих дисциплин?

4.6. Постройте модель с очередями пакетной ПРОП-системы из примера 3.2. Модели с очередями, подобно диаграммам событий, могут рассматриваться как средство описания вычислительных систем. Одной из целей таких описаний является облегчение формулировки имитационных моделей. На основе упомянутого выше примера сравните эти два «языка» (язык моделей с очередями и язык диаграмм событий) в плане их описательной мощности и в особенности в плане удобства их использования при построении имитационных моделей.

4.7. Система памяти состоит из восьми непрослоенных модулей, причем у каждого четыре входа. Запросы на доступ приходят от четырех процессоров, которые связаны с четырьмя независимыми входами каждого модуля. Постройте модель, позволяющую

вычислить среднюю скорость передачи для этой системы памяти, предполагая, что циклы модулей синхронны, что каждый процессор генерирует один запрос за цикл памяти, что запросы генерируются каждым процессором независимо и равномерно распределяются среди модулей и что для разрешения конфликтов на входах каждого модуля используется правило приоритетов, а отвергнутые запросы повторяются в следующем цикле до тех пор, пока они не будут приняты.

4.8. Определите влияние длины кванта q на функцию ответа для круговой дисциплины.

(a) Подойдите к задаче на символьном уровне, имея в виду, что при изменении q среднее время обслуживания задания остается постоянным.

(b) Для значений параметров, заданных в табл. 4.7, нарисуйте начальные участки функций ответа, соответствующих $q=0.15$ с и $q=0.6$ с и сравните их с кривой (c) на рис. 4.11. Какое значение более благоприятно для коротких заданий?

(c) Существует ли, по вашему мнению, оптимальная длина кванта для данной рабочей нагрузки? Если существует, то как бы вы ее могли определить?

4.9. Покажите, каким образом джексоновский подход к исследованию открытых сетей может быть распространен и на замкнутые сети. Исследуйте, например, сеть на рис. 4.18, используя подход Джексона, и покажите, что полученные результаты совпадают с результатами из разд. 4.3.5, которые были получены методом Гордона и Ньюэлла.

4.10. В системе с 20 терминалами каждому терминалу приписан собственный центральный процессор (типа микропроцессора). Все процессоры связаны с временно приписанными модулями памяти через синхронизированную магистраль, ширина полосы пропускания которой в 10 раз больше ширины полосы пропускания процессоров и каждого модуля памяти. Управляющий процессор (того же типа, что и центральные процессоры) обрабатывает страничные отказы, а другой процессор работает как контроллер диска, который по магистрали пересылает запрошенные центральными процессорами страницы в соответствующие модули памяти. Дисковые пересылки имеют больший приоритет на магистрали, чем обращения процессоров к памяти. Сформулируйте модель с очередями для этой системы, исследуйте ее разрешимость и, предполагая, что система существует, опишите эксперименты, которые следовало бы провести для определения параметров и распределений, необходимых для пользования моделью. Обсудите разумность сделанных вами в процессе формулировки приближений.

4.11. Решите модель на рис. 4.20 в предположении, что центр обслуживания системы управляется согласно непрерываемой дисциплине (например, FCFS). Отметим, что возвратный путь вокруг Q_s и Σ_s должен быть удален. Сравните ваши результаты с результатами из примера 4.10. Можете ли вы предложить применение вашей модели?

4.12. Должно быть проведено сравнение производительности мультипрограммной и мультипроцессорной систем. Для простоты рассматривается единственная мера производительности — среднее время ответа (или время прохождения), предполагается что все распределения экспоненциальные, а дисциплина планирования — FCFS. Интенсивность входного потока λ одинакова для обеих систем, а интенсивность обслуживания каждого из n процессоров в мультипроцессорной системе равна μ/n, где μ — интенсивность обслуживания отдельного мультипрограммного процессора.

(a) Определите среднее время ответа и функцию ответа для n-канальной модели с единственным прибором и с одним центром обслуживания, приспособленной для мультипроцессорной системы.

(b) Сравните уравнения, выражающие $\bar{t}$ и $\overline{t(t_s)}$ для рассматриваемых моделей и объясните, что происходит, когда меняется λ или μ.

(c) Проведенное исследование игнорирует накладные расходы системы. Каким образом, как вы ожидаете, изменились бы ваши результаты, если бы накладные расходы можно было учесть?

4.13. Постройте модель сети с очередями для системы, описанной в задаче 4.2.

(a) Объясните, почему эта модель была бы более точным представлением системы, если нам удастся сделать путь, по которому следует задание при уходе из центра обслуживания, зависящим от того, откуда поступило задание в этот центр.

(b) Ответьте, смогли ли бы вы и каким образом достичь цели, поставленной в части (a), если бы знали, как решить такую сеть с очередями при различных классах заданий, в которой задание может изменять свой класс при переходе в новый центр обслуживания [см. Баскетт и др. (1975)].

4.14. Сколько терминалов насыщают диалоговую систему, если среднее терминальное время 20 с, а среднее процессорное время, требуемое пользователем, 0.1? 0.5 с? 1 с? Каково значение среднего времени ответа в точке насыщения в этих трех случаях?

4.15. Включите модель с рис. 4.20 в модель — открытую сеть, в которой задания, появляющиеся из источника, ожидают в очереди к памяти, пока не наступит их черед присоединяться к очереди центрального процессора Q_s. Задания, покидающие обслуживающий прибор центрального процессора Σ_s, переходят с вероятностью b_{st} в центр обслуживания Σ_t, представляющий N параллельных, одинаковых и независимых устройств ввода-вывода. Когда задание покидает систему (это событие происходит с вероятностью $1-b_{st}$), другое задание из очереди к памяти поступает в очередь к центральному процессору, так что число заданий N в этом цикле остается постоянным. Решите модель относительно ее среднего времени ответа, предполагая, что вероятностью того, что очередь к памяти пуста, можно пренебречь. Можно ли считать цикл с двумя центрами обслуживания N-канальным прибором? Или одноканальным? При каких предположениях? Можете ли вы вычислить минимальное λ, при котором вероятность того, что очередь памяти пуста, меньше 0.05? Пусть $b_{st}=0.9$, а значения параметров указаны в табл. 4.11.

4.16. Замените каждый центр обслуживания в открытой сети на рис. 4.16 прямоугольником, изображающим задержку. Рассчитайте эти задержки, используя результаты, полученные для $M/M/1$-модели, и значения параметров из табл. 4.8. Затем вычислите среднее время ответа сети с задержками и убедитесь, что оно равно значению, полученному в примере 4.8.

5
Описание рабочей нагрузки

5.1. ВВЕДЕНИЕ

Практически не существует исследования по оценке производительности вычислительной системы, в котором не встает задача описания рабочей нагрузки. Во многих случаях описание рабочей нагрузки является для исследователя самой трудной технической задачей. Чтобы подчеркнуть важность этой задачи и нашу убежденность в том, что ее наиболее существенные идейные аспекты являются общими для всех исследований и методов оценки, мы посвятим эту главу обсуждению проблемы описания рабочей нагрузки.

Рабочая нагрузка вычислительной системы была определена в разд. 1.2 как совокупность всей входной информации (программ, данных, команд), поступающих в систему извне. Это определение требует некоторых пояснений и оговорок.

Во-первых, состав рабочей нагрузки зависит от выбора исследователем границ системы. Процесс или задание пользователя обычно рассматриваются как внешние по отношению к системе. Они, таким образом, являются частью рабочей нагрузки. Модули операционной системы, управляющие ресурсами, обычно рассматриваются как часть системы. Системные программы, работающие для каждого отдельного пользователя, запросившего их услуги (например, компиляторы, ассемблеры, редакторы, базы данных), составляют «серую область», поскольку их принадлежность системе или рабочей нагрузке зависит от того, какое исследование нужно провести. Во всяком случае, такое разграничение всегда желательно для прояснения концепций и целей исследователя и во избежание распространенных ошибок.

Во-вторых, приведенное выше определение рабочей нагрузки неполно, если не определить точно временные границы, которые будут рассматриваться. Мы можем, например, включить в рабочую нагрузку установки всю входную информацию, обрабатываемую системой на этой установке в течение дня. Назовем это *дневной рабочей нагрузкой*. Таким же образом мы можем определить *часовую*, *недельную*, *месячную* и *годовую рабочие нагрузки*. Есть у нас также *рабочая нагрузка за время существования*: вся входная информация, полученная и обработанная системой за все время ее существования. Другие перечисленные выше типы рабочих нагрузок могут рассматриваться как подмножества рабочей нагрузки за время существования. При уменьшении длительности рассматриваемого периода времени становятся необходимыми дальнейшие уточнения, поскольку возрастает важность граничных

условий. Например, в системе с пакетной обработкой в течение 1-минутного периода некоторые задания считываются, некоторые задания покидают ее, некоторые приближаются к завершению, а другие ожидают на ленте или на диске. Какие из этих заданий должны быть включены в 1-минутную рабочую нагрузку? Ответ зависит от особенностей проводимого исследования. В большинстве случаев в соответствующую рабочую нагрузку включаются только запросы на ресурсы, полностью или частично удовлетворенные в течение данного периода.

В-третьих, не нужно из приведенного выше определения рабочей нагрузки делать вывод, что поведение системы не влияет на ту входную информацию, которую система получает извне. Известно, что коллектив пользователей установки довольно ощутимо и не очень медленно реагирует на ряд системных изменений, но об этом имеется так мало количественной информации, что эти изменения невозможно надежно предсказать. Поэтому обычно делаются упрощающие предположения о том, что рабочая нагрузка нечувствительна ко всем изменениям производительности системы. Такое же допущение принято и в этой книге (если явно не оговорено обратное). Исследователь, конечно, должен помнить, что это допущение оправдывается только нашим сегодняшним незнанием.

При всяком оценочном исследовании мы сталкиваемся с проблемой решения того, при какой рабочей нагрузке (или рабочих нагрузках) должна исследоваться производительность системы. Индексы производительности имеют смысл только в том случае, когда точно определена рабочая нагрузка, при которой эти индексы получены. Даже когда измерения в системе при ее *производственной рабочей нагрузке*, рабочая нагрузка должна до некоторой степени управляться посредством выбора моментов начала и длительности измерений. Экспериментирование в *управляемой среде* или по крайней мере в *наблюдаемой среде* системы является существенным условием *воспроизводимости* эксперимента, т. е. его научной значимости. Эксперимент с вычислительной системой при неуправляемой и ненаблюдаемой рабочей нагрузке — это то же, что измерение электропроводности материала без учета температуры или попытка определить соотношение между давлением и температурой газа, игнорируя его объем.

Есть две операции, относящиеся к рабочей нагрузке, которые должны быть выполнены в течение этапа планирования оценочного исследования. Первая — это *уточнение* типа рабочей нагрузки, которая приводит в действие систему или модель системы. В большинстве исследований это текущая производственная рабочая нагрузка, определенная в течение некоторого периода времени (например, неделя, месяц, год), который тоже должен быть уточнен. Исследователя иногда интересует оценка производительности системы при других типах рабочих нагрузок. Чтобы определить, например, мощность (или максимальную пропускную способность) системы, нужно выбрать рабочую нагрузку, насыщающую систему. Для исследований, целью которых является проверка гипотезы относительно причин неэффективности,

выбор специальной неправдоподобной рабочей нагрузки может дать более быстрые и надежные результаты, чем те, которые могли бы быть получены с помощью производственной рабочей нагрузки. В некоторых предикторных исследованиях рабочая нагрузка может представлять собой долговременный или кратковременный прогноз текущей нагрузки. По соображениям, которые будут пояснены ниже, выбранную рабочую нагрузку назовем *реальной рабочей нагрузкой*.

Другой операцией, которую нужно выполнить, является *описание* рабочей нагрузки исследуемой системы. Рабочая нагрузка может быть описана с различной степенью детализации. Читателю, уже изучившему гл. 2, 3 и 4, нетрудно будет признать, что при оценке производительности реальная рабочая нагрузка должна быть охарактеризована *моделью рабочей нагрузки*. Именно так поступают при имитационных или аналитических исследованиях, в которых модель реальной рабочей

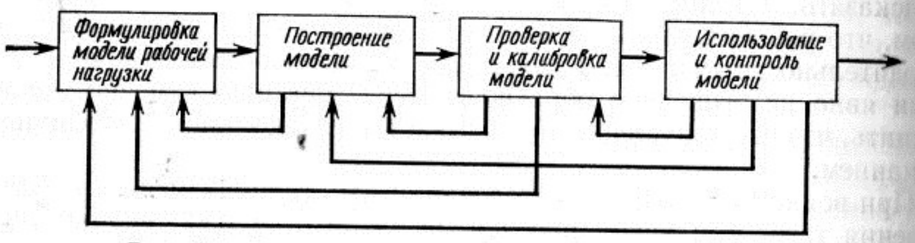


Рис. 5.1. Этапы разработки модели рабочей нагрузки.

нагрузки должна всегда приводить в действие модель реальной системы, если нужно получить значения индексов производительности. То же имеет место и в измерительных экспериментах, только система обычно управляется моделью реальной рабочей нагрузки [см. Феррари (1972)]. В некоторых экспериментах рабочая нагрузка, обрабатываемая системой в течение периода измерений, представляет собой выборку из производственной рабочей нагрузки или из неких ее частей (например, из ее пиковых периодов). Эта выборка, о которой предполагается, что она представляет значительно более обширную реальную рабочую нагрузку, может рассматриваться как модель рабочей нагрузки. В экспериментах других типов система управляется множеством входных данных, сконструированных для измерения этой системы и заменяющих реальную рабочую нагрузку. Эти конструкции, вследствие их происхождения и назначения, вполне могут рассматриваться как модели рабочей нагрузки.

Как и всякая другая модель, модель рабочей нагрузки должна быть сформулирована, построена, оттестирована, откалибрована и проконтролирована. Итеративный процесс создания модели рабочей нагрузки схематически изображен на рис. 5.1.

Проблемы, возникающие при формулировке модели рабочей нагрузки, обсуждаются в разд. 5.2. В разд. 5.3 мы опишем основные приемы, используемые при построении моделей рабочих нагрузок по заданным формулировкам. Калибровка и контроль моделей рабочих нагрузок кратко комментируются в разд. 5.4. Формулировка, построение, проверка, калибровка и контроль описаны по отдельности только для большей ясности. На самом деле они часто бывают запутанно и неявно переплетены.

О модели рабочей нагрузки можно судить по нескольким критериям. Основные характеристики такой модели перечислены в табл. 5.1.

Таблица 5.1

Основные характеристики модели рабочей нагрузки

Представительность
Гибкость
Простота построения
Компактность
Цена использования
Системная независимость
Воспроизводимость
Совместимость

Некоторые из этих характеристик важны для моделей вообще. *Представительность* — это другое выражение для часто используемого в литературе при описании рабочей нагрузки термина *точность*. *Гибкость* — это возможность легкой и недорогой модификации модели, отражающей изменения реальной рабочей нагрузки. *Простота построения* рабочей нагрузки включает стоимость и сложность сбора информации, необходимой для работы с моделью. Такая характеристика, как *компактность*, связана со степенью детализации и, следовательно, с представительностью и ценой использования. Очень компактная модель обычно менее детализирована, менее представительна и дешевле в использовании, чем менее компактная. Важно отметить то, что компактность часто определяется скорее скудностью доступной информации о реальной рабочей нагрузке, чем тем, что отбор информации целиком оставлен на усмотрение пользователя.

Другие характеристики более специфичны именно для моделей рабочей нагрузки. *Системная независимость* — это пределы, в которых модель может быть перенесена с системы на систему, оставаясь при этом достаточно представительной. Типичные задачи, для которых независимость весьма существенна,— это некоторые задачи выбора, рассматриваемые в гл. 6. В других исследованиях требования независимости значительно слабее. Если, например, цель исследования — повышение производительности системы, то модель рабочей нагрузки должна быть независима в смысле приведенного выше определения в пределах модификаций системы, рассматриваемых с целью повышения производительности. Поскольку не все представления рабочей нагрузки, использованные при измерениях, могут быть воспроизведены одинаково легко, важной характеристикой является *воспроизводимость*. Вместе с гибкостью воспроизводимость описывает степень

контроля над рабочей нагрузкой, имеющуюся у исследователя. *Совместимость* с управляемой системой или моделью системы необходима для того, чтобы модель рабочей нагрузки можно было использовать. Это означает, например, что модель, предназначенная для управления реальной системой, должна быть составлена из программ, выполнимых на этой системе.

Две введенные выше модельные характеристики будут исследованы более подробно в разд. 5.2.2 (представительность) и 5.2.3 (независимость). Все эти характеристики рассматриваются при рассмотрении моделей рабочей нагрузки в разд. 5.3.

5.2. ФОРМУЛИРОВКА МОДЕЛИ РАБОЧЕЙ НАГРУЗКИ

5.2.1. Подходы к описанию рабочей нагрузки

Всякую реальную рабочую нагрузку можно рассматривать как состоящую из множества заданий, каждое из которых, когда оно выполняется исследуемой системой, представляет собой некую задачу обработки информации. Описание рабочей нагрузки для целей оценки требует определения тех из множества ее свойств, которые влияют на производительность системы. Так как нас интересуют количественные оценки, то мы рассматриваем модель рабочей нагрузки как множество *параметров рабочей нагрузки*, имеющих количественное выражение. Два примера, почерпнутых соответственно из гл. 3 и 4, будут теперь рассмотрены для пояснения идеи модели рабочей нагрузки и некоторых типов описания, используемых при оценочных исследованиях.

Пример 5.1. В этом примере мы рассмотрим модель рабочей нагрузки, которая управляет моделью ПРОП-системы из примера 3.2. Некоторые представления об этой модели даны в примере 3.4. В табл. 3.6 перечислены параметры, характеризующие рабочую нагрузку имитируемой ПРОП-установки с пакетной обработкой. Подобное описание основывается на предположении, что из всей информации, содержащейся в реальной рабочей нагрузке, только параметры из табл. 3.6 оказывают ощутимое влияние на интересующие нас индексы производительности.

Относительно данной таблицы необходимо сделать некоторые замечания. Первое, модель рабочей нагрузки на самом деле представляет собой совокупность *моделей заданий*, характеризующих каждое задание по отдельности. Это имеет место даже в управляемой распределением имитационной модели, обрабатывающей статистически одинаковые задания, когда каждый параметр задания извлекается из одного и того же распределения. Раздельное описание заданий — очень распространенный подход, так как он проще и естественнее альтернативного метода, заключающегося в непосредственном моделировании рабочей нагрузки в целом, как если бы она порождалась одним единственным заданием. Для такой имитационной модели, какая будет рассматриваться здесь, раздельное описание заданий обязательно, так как истории каждого задания в системе как раз и есть то, что модель пытается воспроизвести.

Другое важное, хотя и очевидное замечание состоит в том, что модель рабочей нагрузки и модель системы должны создаваться вместе. Они должны соответствовать друг другу и быть совместимыми. Степень детализации модели рабочей нагрузки отражает степень детализации имитационной модели, и, как и у последней, ограничена доступностью соответствующих данных. Если, например, длина $r_{дс}$ каждой дисковой записи, управляемой заданием, не известна ни детерминированно, ни статистически, то этот параметр не может служить характеристикой задачи. В этом случае модель диска из табл. 3.5, опирающаяся на $r_{дс}$, не может быть использована.

Для того чтобы быть совместимой с моделью системы, которая в нашем примере представлена своими ресурсами, модель рабочей нагрузки должна быть *ресурсно-ориентированной*. Как видно из табл. 5.2, каждое задание представляется своими запросами на ресурсы оборудования, которые моделируются имитационной моделью.

Таблица 5.2

Описание задания для имитационной модели из примера 3.2*

Ресурс	Общий	Параметры задания: Шаг задания 1	Шаг задания 2	...	Шаг задания n
УЧ	c				
УП	l				
М		$m(1)$	$m(2)$		$m(n)$
ЦП		$t_{цп}(1, 1), B(1, 1)$	$t_{цп}(2, 1), B(2, 1)$	...	$t_{цп}(n, 1), B(n, 1)$
		$t_{цп}(1, 2), B(1, 2)$	$t_{цп}(2, 2), B(2, 2)$	...	$t_{цп}(n, 2), B(n, 2)$
		.	.		.
		.	.		.
		.	.		.
		$t_{цп}(1, n_{цп1})$	$t_{цп}(2, n_{цп2})$	...	$t_{цп}(n, n_{цпn})$
Барабан		$r_{бр}(1, 1)$	$r_{бр}(2, 1)$	...	$r_{бр}(n, 1)$
		.	.		.
		.	.		.
		.	.		.
		$r_{бр}(1, n_{бр1})$	$r_{бр}(2, n_{бр2})$	...	$r_{бр}(n, n_{брn})$
Диск	$r_{дс}$(в)	$r_{дс}(1,$ в)	$r_{дс}(2,$ в)		$r_{дс}(n,$ в)
		$r_{дс}(1, 1)$	$r_{дс}(2, 1)$		$r_{дс}(n, 1)$
		.	.		.
		.	.		.
		.	.		.
	$r_{дс}$ (из)	$r_{дс}(1, n_{дс1})$	$r_{дс}(2, n_{дс2})$	...	$r_{дс}(n, n_{дсn})$
		$r_{дс}(1,$ из)	$r_{дс}(2,$ из)	...	$r_{дс}(n,$ из)

* Обозначения, не разъясненные в примере 5.1, см. в табл. 3.5 и 3.6.

Обозначения в табл. 5.2 должны пониматься следующим образом. Шаг задания j состоит из $n_{цпj}$ интервалов обработки длительностью $t_{цп}(j, i)$ ($i=1, 2, \ldots, n_{цпj}$), за каждым из которых следует или запрос барабана на запись длиной $r_{бр}(j, h)$ ($h=1, 2, \ldots, n_{брj}$), или запрос диска с требованием записи длиной $r_{дс}(j, k)$ ($k=1, 2, \ldots, n_{дсj}$). Двоичный параметр $B(i, j)$ определяет, является ли запрос в конце интервала обработки $t_{цп}(i, j)$ запросом барабана или диска. За последним интервалом обработки шага задания j всегда следует запрос диска с длиной записи $r_{дс}(j,$ из), а первому интервалу предшествует запрос диска с записью длиной $r_{дс}(j,$ в). Длины записей для этих запросов диска, которые обслуживаются устройством чтения с перфокарт в начале выполнения заданий и печатающим устройством в конце выполнения, обозначены соответственно $r_{дс}$(в) и $r_{дс}$(из).

Такое описание каждого задания в терминах ресурсов, нужных для его выполнения, очень естественно и распространено. Оно тем не менее страдает существенным для многих оценочных моделей недостатком: зависимостью модели рабочей нагрузки от системы и ее конфигурации, для которой эта модель предназначена. Другой вопрос, возникающий в связи с моделью, описанной в этом примере, это вопрос о ее представительности. Нет оснований сомневаться в том, что если шесть видов ресурсов, пере-

численных в табл. 5.2, являются теми и только теми ресурсами, которые имеют существенное значение, то вероятно, что эта модель будет точной (при условии, что она правильно построена и откалибрована). Но является ли это обоснованным допущением? ■

Пример 5.2. Рабочая нагрузка модели ПРОП-системы с центральным обслуживающим прибором, изображенной на рис. 4.18, характеризуется параметрами, перечисленными в табл. 4.9 (за исключением параметра N — степени мультипрограммирования). При выборе такого описания мы исходили из предположения, что времена обслуживания для всех центров обслуживания распределены экспоненциально. Более того, система предполагается настолько загруженной, что степень мультипрограммирования всегда находится на уровне максимального значения N. Эта модель рабочей нагрузки может рассматриваться также как совокупность моделей заданий, как и модель из примера 5.1, но здесь задания статистически одинаковы.

Заметим, что времена обслуживания зависят как от запросов заданий, так и от скоростей обслуживающих приборов. Мы считаем, что в нашем исследовании скорости устройств, смоделированных на рис. 4.18 тремя обслуживающими приборами, не будет изменяться. При этом предположении времена обслуживания могут рассматриваться как параметры задания.

Об этой модели можно сделать замечания, очень похожие на те, что были сделаны в примере 5.1. Она, очевидно, совместима с моделью системы и при своей степени детализации с трудом удовлетворяет требованию разрешимости в замкнутой форме. Она ресурсно-ориентирована, хотя моделируются только ресурсы центрального процессора, барабана, диска и до некоторой степени основной памяти. И наконец, способность этой модели правильно и полностью представить реальную рабочую нагрузку еще требуется доказать. ■

В двух предшествующих примерах мы рассматривали модели рабочей нагрузки, сформулированные как совокупности моделей заданий. Модель каждого задания состоит из v-строки $(w_1, w_2, \ldots, w_v)$ *параметров задания*, большинство из которых представляет собой запросы ресурсов для этого задания. Таким образом, рабочая нагрузка — это набор из N v-строк, представляющий N заданий. Наиболее распространенными типами параметров являются *скалярный параметр* и *последовательность*. Последний представляет собой временной ряд скалярных переменных (детерминированных или случайных). Например, модель рабочей нагрузки, рассмотренная в примере 5.1,— это N 7-строк типа $(c, l, \{m\}, \{t_{цп}\}, \{B\}, \{r_{бр}\}, \{r_{дс}\})$, где c и l — скалярные параметры, а $\{m\}$, $\{t_{цп}\}$, $\{B\}$, $\{r_{бр}\}$ и $\{r_{дс}\}$ — последовательности. Упомянутая в примере 5.2 модель задания может рассматриваться как состоящая из четверки $(\{t_1\}, \{t_2\}, \{t_3\}, \{q\})$ последовательностей, где $\{t_i\}$ — последовательность времен обслуживания i-го центра обслуживания, а $\{q\}$ — последовательность троичных переменных, указывающих на центр, в который переходит задание, покидающее прибор Σ_i (центральный процессор). Так как все задания предполагаются статистически одинаковыми, то вся модель рабочей нагрузки может быть представлена как пятерка $(\tau_1, \tau_2, \tau_3, b_{12}, b_{13})$. τ_i — это случайные переменные, распределения которых (в предположении, что это экспоненциальные распределения) полностью определяются их средними $1/\mu_i$. b_{1i} — скаляры, обозначающие вероятности перехода из центра в центр (b_{11}, равный $1-b_{12}-$

b_{13} в модель не включен, поскольку он функционально зависим от остальных параметров рабочей нагрузки).

Подавляющее большинство моделей рабочей нагрузки, используемых при оценочных исследованиях, можно свести к этому типу. В рассмотренных примерах 5.1 и 5.2 мы столкнулись с двумя важными проблемами: проблемой представительности и проблемой системной независимости. Эти проблемы обсуждаются далее в разд. 5.2.2 и 5.2.3.

5.2.2. Проблема представительности

Первостепенным требованием к любой модели является требование представить с удовлетворительной точностью поведение моделируемой системы. Представительность модели рабочей нагрузки определяется ее способностью адекватно представить реальную рабочую нагрузку для исследования по оценке производительности.

По нашей терминологии *реальная* рабочая нагрузка не обязательно должна совпадать с производственной. Представительность модели рабочей нагрузки должна пониматься как точность представления не производственной, а реальной рабочей нагрузки (которая вовсе не обязана существовать физически). Для некоторых исследований полезнее модель рабочей нагрузки, которая заставляет все части системы работать в типичных режимах, чем модель, пытающаяся воспроизвести поведение какой-то специфической производственной нагрузки. Представительность, введенная (согласно предшествующему определению) как способность модели нагрузки удовлетворять требованиям исследователя, оценивающего систему, является одним из важнейших конструктивных соображений даже в таких исследованиях.

Модель рабочей нагрузки обычно компактнее, чем представляемая ею реальная рабочая нагрузка, т. е. она содержит меньше информации. Главным основанием для построения компактных моделей является экономия времени и денег. Как определить, является ли внесенная в модели информация существенной для представительности модели? Может ли представительность модели иметь количественное выражение? В поиске ответов на эти вопросы мы вынуждены признать, что представительность может рассматриваться как частный случай *эквивалентности рабочей нагрузки*.

Рассмотрим ситуацию на рис. 5.2. Индексы производительности, выбранные для определения системы $\mathcal{S}$, в совокупности обозначены символом $\mathcal{P}$. В множество $\mathcal{P}$ могут входить распределения случайных величин, средние, дисперсии и более высокие моменты, значения детерминированных переменных (например, коэффициентов использования) и т. д. Назовем рабочие нагрузки W_1 и W_2 *эквивалентными относительно системы $\mathcal{S}$ и множества индексов производительности $\mathcal{P}$*, если при управлении системой значения выдаваемых ими индексов производительности совпадают (в некоторых определенных пределах точности). Пусть W_2 — это модель нагрузки W_1 и управляет или

системой $\mathscr{S}$ или пригодной моделью системы $\mathscr{S}$. Тогда рабочая нагрузка W_2 *представляет* W_1 *относительно системы* $\mathscr{S}$ *и множества* $\mathscr{P}$, если W_2 эквивалентна W_1 относительно $\mathscr{S}$ и $\mathscr{P}$.

Нельзя не заметить зависимости этого определения от представительности системы и от индексов производительности. Даже если W_2

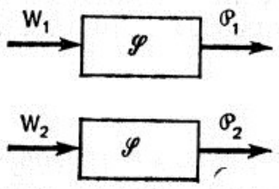


Рис. 5.2. Иллюстрация эквивалентности рабочих нагрузок. W_2 эквивалентна W_1, если $\mathscr{P}_1 = \mathscr{P}_2$.

представляет W_1 для данных $\mathscr{S}$ и $\mathscr{P}$, не следует удивляться, если две одинаковые рабочие нагрузки не будут эквивалентны для разных индексов производительности или для разных систем. Существуют изменения, которые не влияют на эквивалентность или представительность. Это, например, случай, когда распределение из $\mathscr{P}$ заменяется его средним или парой (среднее, дисперсия). Обратное преобразование, однако, не сохраняет эквивалентности рабочих нагрузок. Рассматривая индексы, которые хорошо представляются случайными величинами, можно определить, что две рабочие нагрузки эквивалентны до n-го порядка, если они дают одинаковые значения для по крайней мере первых n моментов распределений этих индексов. Если две рабочие нагрузки эквивалентны до n-го порядка относительно некоторой системы и множества индексов, то они эквивалентны и до i-го порядка ($i=1, 2, \ldots, n-1$) относительно той же системы и того же множества индексов. Можно сделать полезное расширение этого определения, если выбрана некая метрика, дающая количественное «расстояние» между $\mathscr{P}_1$ и $\mathscr{P}_2$. Это позволит нам определить *степень эквивалентности* между W_1 и W_2, или *степень представительности* W_2 как модели W_1.

Перед тем как продолжить наше исследование, применим данное выше определение к модели из примера 5.2.

Пример 5.3. Предполагая, что модель с центральным прибором на рис. 4.18 является достоверной моделью системы $\mathscr{S}$, мы хотим определить условия, при которых две нагрузки W_1 и W_2 эквивалентны относительно $\mathscr{S}$ и множества индексов производительности $\mathscr{P}=(\bar{T}, \bar{t})$. В разд. 4.3.5 мы получили следующие результаты для двух наших индексов производительности:

$$\bar{T} = b_{11}\mu_1\rho_1, \tag{5.1}$$

$$\bar{t} = \frac{N}{b_{11}\mu_1\rho_1}, \tag{5.2}$$

и из них видно, что ρ_1 — коэффициент использования процессора (ЦП) может быть представлена как функция от параметров рабочей нагрузки и от N, как показано в (4.106) и (4.101). Таким образом, две рабочих нагрузки, у которых параметры, приведенные в табл. 4.9, имеют одинаковые значения, являются эквивалентными относительно $\mathscr{S}$ и $\mathscr{P}$. Это условие необходимо, но не достаточно. Два необходимых и до-

статочных условия эквивалентности рабочих нагрузок W' и W'' это

$$b'_{11}\mu'_1\rho'_1 = b''_{11}\mu''_1\rho''_1, \quad N' = N''. \tag{5.3}$$

Если равенства (5.3) выполняются, то W'' эквивалентна W' и может использоваться для управления $\mathscr{S}$ вместо нее. Ясно, что различия, имеющиеся между W_1 и W_2 и не отраженные их параметрами, могут сыграть важную и нежелательную роль при внесении изменений в $\mathscr{S}$ или в $\mathscr{P}$. Тот факт, например, что первое условие из (5.3) выполняется для определенных значений b'_{1j}, b''_{1j}, μ'_j, μ''_j ($j=1, 2, 3$), не гарантирует, что оно будет выполняться и тогда, когда изменятся скорость центрального процессора или барабана, или диска, или когда изменение распределения файлов повлечет изменение значений b'_{12}, b''_{12}, b'_{13}, b''_{13}. Изменение скорости одного центра обслуживания не повлияет на эквивалентность двух рабочих нагрузок, у которых значения всех параметров из табл. 4.9 совпадают. Как было замечено выше, это не являющееся необходимым условие значительно сильнее (5.3). Таким образом, можно ожидать, что пригодность модели рабочей нагрузки возрастает по крайней мере до некоторого этапа, когда растет степень детальности приближения этой модели к реальной рабочей нагрузке. ■

Наше определение представительности основывается на эквивалентности рабочих нагрузок, и, следовательно, на индексах производительности, выбранных для проводимого оценочного исследования. Легко заметить концептуальное сходство между этим определением и критерием, иногда используемым для определения *нагрузки* данной системы. Это сходство состоит в том, что рассматривается коэффициент использования устройств, входящих в конфигурацию системы. Существенно, что этот критерий замеряет *величину* рабочей нагрузки в зависимости от того, какое влияние оказывает эта рабочая нагрузка на деятельность системных устройств. Другими словами, этот критерий характеризует *внешнюю нагрузку* в терминах производимых ею *внутренних нагрузок* различных устройств.

Величина рабочей нагрузки, обрабатываемой системой за определенное время, часто приблизительно оценивается реактивностью системы на внешние воздействия, на которые система реагирует известными способами при различных рабочих нагрузках. Если время прохождения задания или время ответа на команду значительно больше обычного, первое, что предполагает пользователь,— это то, что система загружена много больше обычного. Каруш (1969) предложил метод измерения величин нагрузок диалоговых установок, который основывается на этом наблюдении. Должна быть написана программа, называющаяся *терминальный тест*. Затем терминальный тест запускается с терминала в то время, как система обрабатывает некоторое количество (изменяющееся от нуля до заданного максимума) копий этого теста. Времена ответов для такого теста, получаемые на терминале, измеряются и изображаются на графике, как на рис. 5.3. Когда должна быть измерена величина системной нагрузки, экспериментатор вводит с терминала тест и измеряет время ответа. По этому времени на диаграмме рис. 5.3 можно определить значение рабочей нагрузки в системе во время эксперимента, выражен-

ное соответствующим числом терминальных тестов. Тест, таким образом, используется в качестве единицы рабочей нагрузки.

Предложенный Симсом (1974) *фактор нагрузки* диалоговой системы тоже основывается на идее эквивалентности. Все смеси команд, одновременно поступающие на системные терминалы, дают одно и то же среднее значение времени ответа, соответствующее одному и

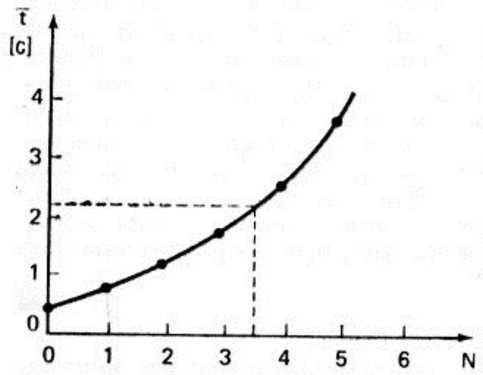


Рис. 5.3. Терминальный тест для измерения величины рабочей нагрузки диалоговой системы. Среднее время ответа 2.2 с соответствует нагрузке в 3.4 терминального теста.

тому же фактору нагрузки. Введение фактора нагрузки излишне, так как выбранный индекс производительности (в данном случае — среднее время ответа) может непосредственно использоваться как количественное выражение величины нагрузки. Значительно полезнее выразить, если это возможно, фактор нагрузки L (пропорциональный среднему времени ответа) как функцию от количеств терминалов, с которых поступают различные типы команд. И вообще, если мы рассматриваем k команд $c_1, \ldots, c_k$ и N терминалов, то нагрузку системы можно определить как k-строку $N_1, \ldots, N_k$, где N_i $(i=1, \ldots, k)$ — число терминалов, с которых поступает команда c_i, а $\sum_{i=1}^{k} N_i = N$. Для некоторых диалоговых систем Симсу (1974) удалось найти линейное выражение вида

$$L = L(N_1, \ldots, N_k) = \sum_{i=1}^{k} a_i N_i. \qquad (5.4)$$

Более подробно работа Симса рассматривается в разд. 6.3.5 и 6.5.

Определение представительности, основанное на эквивалентности рабочих нагрузок, позволяет нам сформулировать критерий для установления степени детализации модели рабочей нагрузки и параметров, которые должны быть в нее включены. Для данной системы $\mathscr{S}$, для данного множества индексов производительности $\mathscr{P}$ и для данной реальной рабочей нагрузки W в W' (модель W) должен быть включен такой и только такой параметр из W, влиянием которого на некий элемент из $\mathscr{P}$ нельзя пренебречь. Интересующее нас влияние — это зависимость некоего индекса из $\mathscr{P}$ от значений параметров рабочей нагрузки в предполагаемых пределах изменения этих параметров для некой рабочей нагрузки (или всех рабочих нагрузок), рассматриваемых в данном исследовании.

Насколько просто найти описание, удовлетворяющее критерию представительности? В целом эта проблема была рассмотрена в разд. 2.6 и 2.7, где были введены некоторые методы для определения влияния параметров системы и рабочей нагрузки на заданные индексы производительности. Часто зависимости индексов производительности от наиболее очевидных параметров рабочей нагрузки (например, от запросов на ресурсы заданий) являются существенно нелинейными. Таким образом, эти зависимости обычно неаддитивны, а взаимодействиями между параметрами рабочей нагрузки, так же как и взаимодействиями между параметрами системы и рабочей нагрузки, пренебрегать нельзя. Наличие взаимодействий редко рассматривается при формулировке рабочей нагрузки. Выбор параметров задания, не отражающих существование этих взаимодействий, может оказаться неподходящим. Например, модель рабочей нагрузки, не содержащая информации о порядке появления заданий, может не давать адекватного представления о рабочей нагрузке. Это может, среди прочего, объясняться тем, что в мультипрограммной системе значения большинства индексов производительности (например, средней пропускной способности, использования центрального процессора и устройств ввода-вывода) обычно очень чувствительны к такому упорядочиванию.

5.2.3. Проблема системно-независимого описания

Большинство практически интересных исследований предполагает сравнение различных систем или различных модификаций одной системы. Поэтому независимость — это важная характеристика модели рабочей нагрузки. Эта характеристика существенно зависит от способа формулировки задачи.

В разд. 5.1 было замечено, что рассматривать поведение коллектива пользователей независимо от производительности системы, которой они пользуются, вообще говоря, неправильно, но (из-за уровня наших знаний) это необходимое допущение. Вид системной зависимости, которую мы рассмотрим в данном разделе, совсем иного рода. Если параметры, с помощью которых мы моделируем рабочую нагрузку, зависят от системы или от ее конфигурации, или от ее программного обеспечения, то они описывают рабочую нагрузку не так, как должно. Описанную таким образом (например, в терминах запросов на ресурсы) рабочую нагрузку нельзя в неизменном виде *перенести* на другие системы или даже на ту же самую систему со слегка измененной конфигурацией. Вот три простых примера таких ситуаций.

а) Модель рабочей нагрузки, в которой доступ к файлам представлен суммарным временем обращения к диску для каждого задания, включая время ожидания для канала и для диска, не может быть использована для исследования системы с двумя различными

размещениями файлов на диске или с другим числом дисков, или с другой производительностью дисков, или с другой стратегией планирования для дисков.

b) Когда модель рабочей нагрузки характеризует производственную активность задания в терминах запрошенного им процессорного времени вместо, скажем, частот использования команд, то, не внося изменений, нельзя использовать эту модель для оценки системы с центральным процессором того же типа, но с другой скоростью.

c) Модель рабочей нагрузки, в которой работа каждого задания со страницами виртуальной памяти характеризуется общей частотой страничных отказов в процессе счета, бесполезна, если одним из исследуемых факторов является размер основной памяти.

Когда модель рабочей нагрузки представляет собой совокупность моделей заданий, дополнительным источником системной зависимости является выбор такой модели задания, которая в чем-то зависит от других заданий. Это, например, случай, когда в дисковые времена заданий входит ожидание в очереди.

Другим, менее очевидным типом зависимости является зависимость из-за взаимодействий системы и рабочей нагрузки, не учтенных в модели нагрузки. Пример такого взаимодействия — связь между упорядочением, которому подчиняются задания в мультипрограммной системе, и системной стратегией планирования. Пусть некая модель рабочей нагрузки состоит из определенного числа моделей заданий, имеющих определенное упорядочение, адекватно представляющее заданную реальную нагрузку для мультипрограммной системы $\mathscr{S}_1$. Относительно системы $\mathscr{S}_2$, отличающейся от $\mathscr{S}_1$ стратегией планирования, эта модель нагрузки с тем же упорядочением не будет, вероятно, представительной для данной модели рабочей нагрузки. Из-за наличия взаимодействий при описании рабочей нагрузки возникают очень тонкие проблемы. Системные зависимости, обусловленные взаимодействиями, довольно трудно обнаружить и устранить. В принципе все эффекты взаимодействия, которые будут, возможно, важны для определенного исследования, должны быть учтены при построении для этого исследования модели рабочей нагрузки. На практике такой систематический подход может оказаться столь сложным и трудоемким, что конструкторам придется положиться на свою интуицию и на результаты отдельных быстрых неформальных проверок.

Следствия двух типов системной зависимости, рассмотренных в начале раздела, схематически изображены на рис. 5.4. Этот рисунок относится к исследованиям по усовершенствованию, связанным со сравнением двух версий $\mathscr{S}_1$ и $\mathscr{S}_2$ одной и той же системы. $\mathscr{S}_1$ — это существующая, измеренная версия, в то время как $\mathscr{S}_2$ — версия, содержащая модификации, которые еще не могут быть сделаны на реальной системе и влияние которых на производительность нужно предсказать. В сущности мы будем сравнивать $\mathscr{P}_1$ и $\mathscr{P}_2$, т. е. значения индексов производительности, выдаваемых $\mathscr{S}_1$ и $\mathscr{S}_2$ при двух рабочих нагрузках W_1 и W_2, которые будут генерироваться одним коллективом пользователей U, использующих $\mathscr{S}_1$ и $\mathscr{S}_2$ [см. рис. 5.4(a)]. Так как нам неизвестно, как коллектив пользователей будет реагировать на изменения системы, которые переводят $\mathscr{S}_1$ в $\mathscr{S}_2$, то мы не

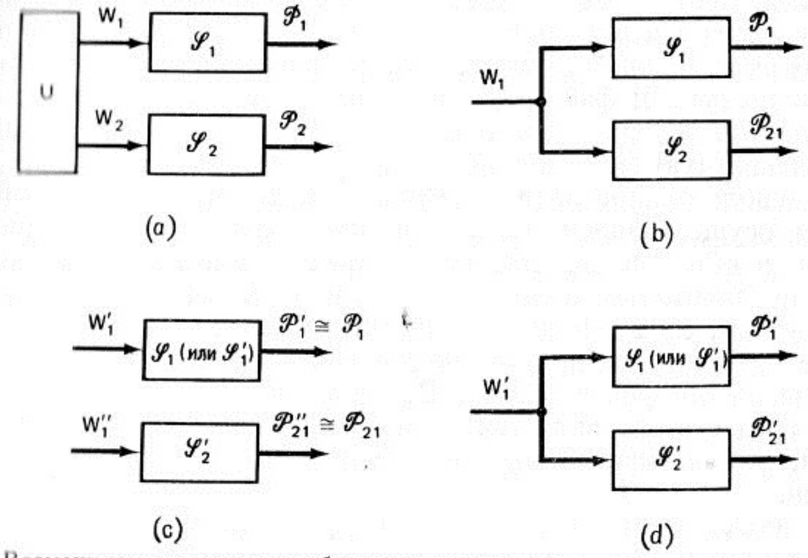


Рис. 5.4. Возможные источники ошибок из-за системной зависимости моделей рабочей нагрузки в исследованиях по усовершенствованию.

можем предсказать W_2, зная W_1. Поэтому мы предполагаем, что и $\mathscr{S}_2$ также будет обрабатывать W_1, давая производительность $\mathscr{P}_{21}$ [см. рис. 5.4(b)]. Реальная модель рабочей нагрузки W_1 будет представлена в исследовании моделью W_1', которая будет использоваться для управления или системой $\mathscr{S}_1$, или ее пригодной моделью $\mathscr{S}_1'$. Та же модель W_1' может управлять и $\mathscr{S}_2'$, пригодной моделью $\mathscr{S}_2$, если W_1' основывается на описании, независимом от рассматриваемых модификаций. Если используется системно-зависимое описание, должны быть разработаны две отдельные модели W_1, W_1' для $\mathscr{S}_1$ и W_1'' для $\mathscr{S}_2$, чтобы получить производительности, близкие соответственно к $\mathscr{P}_1$ и $\mathscr{P}_{21}$ [см. рис. 5.4(c)]. Однако исследователи часто стараются использовать решение, представленное на рис. 5.4(d). Это решение состоит в использовании одной и той же модели W_1' для управления и $\mathscr{S}_1$ (с результатом $\mathscr{P}_1'$) и $\mathscr{S}_2$ (с результатом $\mathscr{P}_{21}'$). Если пренебречь ошибками из-за неточности моделей, то мы видим два соображения, по которым $\mathscr{P}_{21}'$ отличается от $\mathscr{P}_2$: во-первых, потому что W_1 отличается от W_2 из-за чувствительности коллектива пользователей к изменению производительности системы и, во-вторых, потому что W_1' отличается от W_1'' из-за системно-зависимого описания рабочей нагрузки. Перед тем как принять решения, основанные на сравнении $\mathscr{P}_1'$ и $\mathscr{P}_{21}'$, исследователь должен определить влияние этих ошибок

на значения индексов производительности и посмотреть, имеет ли смысл это сравнение.

Существуют ли системно-независимые способы описания рабочей нагрузки? Параметры ресурсно-ориентированных моделей рабочих нагрузок могут в различной степени зависеть от системы. В одном из случаев, кратко упомянутых в начале этого раздела, нужно было исследовать изменения производительности из-за различных способов размещения файлов на диске. Мы установили, что представление способов доступа к файлу для определенной рабочей нагрузки, выраженное в общем дисковом времени на задание, является неадекватным. Адекватную модель рабочей нагрузки можно получить, представив использование заданием файлов в терминах имен файлов, к которым осуществлялся доступ из каждого задания, истории этих доступов и объемов данных, переданных за один доступ. Это представление значительно более подробно, чем то, которое было признано неадекватным, но в данном случае это не важно. На самом деле некоторые из перечисленных выше данных часто могут быть опущены с небольшой потерей точности. Важным преимуществом нового описания перед старым является то, что оно основывается на запросах задания, выраженных в терминах скорее *логических*, чем *физических* ресурсов.

Чем более системно-независимы ресурсы, запросы на которые используются в модели рабочей нагрузки, тем более системно-независимой будет модель рабочей нагрузки. Однако степень независимости неизбежно изменяется от исследования к исследованию. Для исследования по усовершенствованию, в котором не рассматриваются изменения дисковой системы, разобранная выше модель может оказаться адекватной.

При описании рабочей нагрузки для некоторого исследования должны быть внимательно рассмотрены его цели и масштаб. Те характеристики рабочей нагрузки, что связаны с частями системы (или систем), которые в течение исследования предполагается изменять, должны быть представлены по возможности на логическом уровне. Пусть целью исследования является выбор наиболее подходящего для некоторой установки центрального процессора в рамках семейства вычислительных машин в предположении, что все они имеют одинаковый набор команд. Тогда вычислительные запросы заданий могут быть более адекватно представлены количествами различных типов команд, которые нужно выполнить, чем процессорными временами, запрошенными заданиями при выполнении на одном из процессоров из данного семейства.

Альтернативный подход — использование системно-зависимых параметров и соответствующее изменение их значений при переходе с одной системы на другую. Этот подход, изображенный на рис. 5.4(с), применим, только если известен способ преобразования W_1' в W_1''. Предположим, что в примере из предыдущего абзаца отношение между временами выполнения команд каждого процессора и соответствующими временами любого другого процессора одинаково для всех команд. Тогда временные запросы задания для всех центральных процессоров могут быть легко вычислены по временному запросу для одного процессора. Иногда это делается, даже если отношения времен выполнения команд неодинаковы и даже когда различаются наборы команд. Для этого используют оценки относительных скоростей центральных процессоров для рассматриваемых приложений. Ясно, что это грубое приближение, и его пределы достоверности должны быть тщательно исследованы, прежде чем на них можно будет положиться.

В общем изменение параметров рабочей нагрузки при переносе модели рабочей нагрузки на другую систему становится исключительно сложным, если имеются системно-зависимые взаимодействия между этими параметрами. Для выявления, классификации и количественного выражения этих взаимодействий и по возможности определения относительно независимых параметров рабочей нагрузки (равно как и характеристик системы) необходим значительный объем исследований.

Проблема системной зависимости исчезнет, если потребности коллектива пользователей в обработке данных могут быть выражены совершенно абстрактно, без каких-либо ссылок на то, какими средствами эти потребности удовлетворяются на конкретной установке. Это абстрактное описание должно, однако, легко переводиться в более конкретное (например, в терминах запросов на ресурсы), как только управляемая система будет определена. В этом направлении можно предложить, как сказано выше, использовать по возможности широко логические модели, а не физические модели, основывающиеся на ресурсах.

Некоторые исследователи предлагали подходы к абстрактному описанию нужд по обработке информации — фундаментально важной проблемы машинной математики. Основной целью большинства таких попыток (они будут сейчас коротко разобраны) было дать количественное определение *трудоемкости обработки информации*, необходимой процессу. Как замечено Хеллерманом (1972), обычный подход состоит в выборе *канонической реализации* алгоритмического процесса и в определении трудоемкости этого процесса как некоего количественного свойства его канонической реализации.

Канонические реализации, используемые в теории сложности вычислений, представляют собой более или менее абстрактные машины, а свойства, определяющие трудоемкость,— пространственные характеристики (длину ленты для машины Тьюринга, число ячеек памяти, число логических элементов) и временные характеристики (число шагов, число сложений или умножений, максимальное число итераций). Эти подходы обычно предназначены для анализа алгоритмов. Практической целью является определение наиболее эффективного алгоритма среди реализующих одну и ту же функцию. Наибольшая эффективность соответствует минимальному объему работы. Анализ

алгоритмов чрезвычайно полезен при построении программ и систем. Однако определение трудоемкости, вводимое при анализе алгоритмов, хоть и системно-независимо, не оказывает существенной помощи для практической оценки производительности. Это происходит, вероятно, из-за преимущественной ориентации только на вычислительные аспекты (пересылки информации, потребляющие много системных ресурсов, обычно не учитываются этими определениями) и из-за отсутствия методов перевода оценок трудоемкости в ресурсные запросы для заданной системы.

Другие исследования предлагают определения трудоемкости, основывающиеся на теоретико-информационных идеях. Хеллерман (1972), например, определяет *трудоемкость процесса* f: $X \to Y$, чья область определения X конечна и может быть разбита на n подобластей X_i ($i=1, \ldots, n$), таких, что все входные данные из X_i дают одну и ту же точку из области Y, следующим образом:

$$w(f) = \sum_{i=1}^{n} |X_i| \log_2 \frac{|X|}{|X_i|}, \tag{5.5}$$

где $|A|$ — число элементов множества A. Читатель, знакомый с основами теории информации, заметит, что $w(f)/|X|$ — *энтропия* источника X.

Байю (1973) отличает *обратимые преобразования* информации, которые могут только изменять порядок входных элементов, от *необратимых*, при которых $|X| \neq |Y|$. Он определяет *деятельность* процесса f следующим образом:

$$a(f) = \frac{1}{|X|} \sum_{i=1}^{n} |X_i| \log_2 |X_i|, \tag{5.6}$$

что соответствует потере информации при вводе, если f необратима, и потере упорядоченности, если f обратима. В последнем случае подобласти X_i являются наибольшими, и они упорядочены согласно некоторому условному определению порядка. В отличие от подхода Хеллермана в этом определении не учитывается деятельность, связанная с идентичными преобразованиями, т. е. с пересылкой данных.

Источники информации и их энтропии рассматриваются также Розвадовски (1973) в его определении *количества вычислений* как суммарного уменьшения неопределенностей из-за всех (информационно-теоретических) источников, составляющих процесс. Например, для процесса выполнения одной машинной команды, ссылающейся на память, определяются три источника: источник S_1, выбирающий команду для выполнения, производимую операцию и используемые регистры, а также определяющий адрес следующей команды; источник S_2, выбирающий значения операндов, и источник S_3, выбирающий значение результата. Пусть у машины память $32K$ слов, длина слова 24 бит, 8 центральных регистров и длина кода операции 6 бит. Умень-

шения неопределенностей из-за перечисленных выше источников S_1, S_2, S_3 при выполнении команды, осуществляющей сложение содержимого ячейки памяти с содержимым центрального регистра, будут (если все выходы при выборе равновероятны) такими:

$$\begin{aligned} H(S_1) &= \log_2 32K + \log_2 2^6 + \log_2 8 + \log_2 32K = 39 \text{ битов}, \\ H(S_2) &= \log_2 2^{24} + \log_2 2^{24} = 48 \text{ битов}, \\ H(S_3) &= \log_2 2^{24} = 24 \text{ бита}. \end{aligned} \tag{5.7}$$

Таким образом, совокупная трудоемкость, или количество вычислений, произведенное командой, равно

$$H(S_1) + H(S_2) + H(S_3) = 111 \text{ битов}, \tag{5.8}$$

что совпадает с общим числом битов, перемещенных машиной в течение цикла команды. Произведенная S_1 работа может рассматриваться как накладные расходы или расходы по управлению, а работа S_2 и S_3 как действительная, полезная работа.

Определение трудоемкости *работы программного обеспечения*, предложенное Коленсом в контексте его *физики программного обеспечения* [см. Коленс (1972)] с практической точки зрения очень похоже на определение Розвадовски: «Процессор производит одну единицу программной работы в некой запоминающей среде, если изменяется один байт этой среды». Заметим, что и простая пересылка требует работы, даже если пересылаемая информация на самом деле не отличается от имеющейся в памяти до пересылки. Таким образом, термин «изменяется» нельзя понимать буквально. Слово «процессор» в предыдущем определении означает не только центральные процессоры, а все виды процессоров, включая каналы, контроллеры, устройства управления и т. д.

Этот очень краткий и поверхностный обзор подходов к проблеме системно-независимого определения трудоемкости работы по обработке информации предназначался для того, чтобы дать читателю общее представление о подобных подходах и разжечь его любопытство. Даже из нашего краткого описания, вероятно, легко вынести впечатление о том, что все эти подходы или слишком абстрактны или слишком еще системно-зависимы с точки зрения наших целей. Наша идеальная модель рабочей нагрузки достаточно обособлена от реальной системы, так что на нее не влияют любые системные особенности, и тем не менее она легко приспосабливаема к любой системе и ее особенностям. Утверждение о том, что предложенные пока подходы не годятся для решения нашей проблемы, не означает, что они не должны быть изучены, поняты и развиты. Если наше идеальное описание рабочей нагрузки может быть найдено, то имеется шанс, что это будет результатом исследований в описанных выше направлениях. И даже если этот шанс невелик, такие исследования были бы оправданы уже тем, что они прояснили бы природу и основания обработки информации.

5.3. ПОСТРОЕНИЕ МОДЕЛЕЙ РАБОЧИХ НАГРУЗОК

5.3.1. Основные определения

Для представления рабочей нагрузки установки при оценке производительности используется множество различных моделей. Выбор наиболее подходящей модели ограничен тем, что модель рабочей нагрузки должна быть совместима с управляемой ею системой (или моделью системы). Другие существенные ограничения часто возникают из-за ограниченности доступной информации о моделируемой реальной рабочей нагрузке.

Когда нужно построить модель рабочей нагрузки, необходимо внимательно рассмотреть цели исследования и требования к нему в терминах, перечисленных в табл. 5.1. В нескольких последующих разделах мы опишем наиболее важные типы моделей и их свойства. По-видимому, в большинстве исследований рассмотрение моделей ограничивается вводимыми ниже типами моделей.

Спектр характеристик существующих типов моделей рабочих нагрузок довольно широк и не поддается простой классификации. Мы будем основывать нашу классификацию на характере модели. Если модель — это просто выборочный поток заданий, взятый из производственной рабочей нагрузки и использованный для управления системой в течение того же времени, за какое он выдается коллективом пользователей, то мы назовем его *естественной моделью рабочей нагрузки*. Построение и свойства моделей этого типа, которые по своему определению могут использоваться только в измерительных экспериментах, рассматриваются в разд. 5.3.2. Во всех других случаях мы имеем *искусственную модель рабочей нагрузки*. Если искусственная модель состоит из одной или более программ, то она называется *исполняемой искусственной моделью*. Исполняемые модели используются для управления реальными системами в измерительных экспериментах, когда система предоставляется для этих экспериментов. Однако в принципе они могут использоваться в программно-управляемых моделях. Характеристики исполняемых искусственных моделей обсуждаются в разд. 5.3.5. Свойства *неисполняемых искусственных моделей*, применяющихся в аналитических и в большинстве имитационных исследований, кратко рассматриваются в разд. 5.3.3.

5.3.2. Естественные модели рабочих нагрузок

Согласно определению, данному в разд. 5.3.1, естественная модель — это выборка из производственной рабочей нагрузки, выполняемая системой во время проведения эксперимента. Мы называем это моделью реальной рабочей нагрузки только из-за продолжительности, которая обычно меньше, чем у моделируемой нагрузки. Организация моделирования в этом случае состоит только в выборе времен, в которые должен начинаться и кончаться ввод совокупности данных.

Иногда реальная рабочая нагрузка соответствует не производственной нагрузке, а скорее некоторым ее частям. Если мы интересуемся, скажем, поведением системы при большой нагрузке, то нашу годовую рабочую нагрузку можно определить как множество заданий, выполненных системой между 9 час. 30 мин. и 11 час. 30 мин. за все рабочие дни года.

Таким образом, наша модель рабочей нагрузки может состоять из заданий, выполненных в один и тот же интервал суток в течение специально отобранной недели. Заметим, что мы исключили из класса естественных моделей рабочих нагрузок такие выборки из производственных нагрузок, которые запоминаются и позднее используются для управления системой (или ее подходящей моделью). Такие модели считаются искусственными и описываются в разд. 5.3.3—5.3.5.

В литературе сообщается о многих исследованиях, использующих естественную модель рабочей нагрузки, но лишь в немногих из них принятые для модели конструкторские решения описаны и удовлетворительно обоснованы. Шерр (1967) собрал данные о поведении CTSS, диалоговой системы, созданной в Массачусетском технологическом институте, и данные о коллективе пользователей в течение 47 периодов измерений, продолжительность которых в среднем составляла 143 мин. и изменялась от менее чем 1 ч. до примерно 8 ч. Коллектив пользователей характеризуется двумя параметрами — временем обдумывания и временем центрального процессора, запрашиваемым на одно взаимодействие. Среднее квадратичное отклонение средних для этих параметров составляет приблизительно 15% общих средних значений. Этот результат, к сожалению, не гарантирует, что выбранная естественная модель является представительной. Метод независимых копий позволяет нам вычислить по данным из табл. 5.3 при заданном уровне значимости доверительные интервалы оценок, приведенных в первой колонке таблицы, для среднего времени

Таблица 5.3

Характеристики естественной модели рабочей нагрузки для измерения производительности системы CTSS

Параметр рабочей нагрузки	Время обдумывания	Время ЦП/взаимодействие
Общие средние (с)	35.0	0.88*
Медиана 47 средних (с)	34.0	0.93
Среднее 47 средних (с)	34.2	0.92
Среднее квадратичное отклонение 47 средних (с)	4.994	0.150
Коэффициент вариации	0.146	0.163

Источник: Шерр (1967).
* Значение вычислено по данным, собранным в обычные рабочие часы. Все другие позиции содержат результаты вечерних и воскресных измерений.

обдумывания и для среднего процессорного времени, запрашиваемого на одно взаимодействие. Однако можно было бы гарантировать представительность выбранного описания рабочей нагрузки, только если бы мы располагали аналогичной таблицей со значениями индексов производительности. Получаемая нами из табл. 5.3 информация состоит в том, что средние параметров двух рабочих нагрузок являются довольно устойчивыми относительно временных интервалов, в которые измеряется система, и даже за такой короткий период, как 1 ч, дают приемлемую точность оценки этих средних. Таким образом, мы можем предположить, что предложенное Шерром описание рабочей нагрузки является представительным и что его реализация модели, которая была описана выше, дает представительную естественную модель рабочей нагрузки. Тем не менее приведенные в табл. 5.3 и в наших рассмотрениях данные не дают никакой информации о представительности этого описания. Это, конечно, не доказывает, что описание Шерра вообще не представительно.

Из этого примера мы можем заключить, что модель рабочей нагрузки адекватно представляет реальную нагрузку тогда и только тогда, когда и ее формулировка, и ее реализация являются представительными. Потенциально представительная формулировка может дать непредставительную модель, если ее реализация неадекватна. Точно так же потенциально представительный метод построения может дать непредставительную модель, если он использовал неадекватное описание рабочей нагрузки.

В случае естественных моделей их формулировки можно избежать. Если параметры рабочей нагрузки, влияющие на индексы производительности, предполагаются стационарными, то нет необходимости их определять и измерять. Однако измерение некоторых параметров рабочей нагрузки, как это делает Шерр, позволяет нам определить величину и колебания нагрузки и увидеть, насколько ее поведение отличается от стационарного. К тому же описание рабочей нагрузки с помощью измеримых параметров делает эксперименты более воспроизводимыми и делает возможной, если требуется, статистическую интерпретацию результатов (см. пример 2.11).

Построение естественной модели (в предположении стационарности) может основываться на описанных в разд. 2.6.4 методах выбора времени и длительности серии измерений. Когда серия состоит из некоторого числа отдельных относительно коротких подсерий, как в методе независимых копий, то для оценки изменчивости должны быть также учтены замечания относительно конечных условий, сделанные в разд. 3.4.2.

Возможность приложить методы из разд. 2.6.4 появляется из-за того, что проблема *сходимости по вероятности*, рассматриваемая в том разделе, и проблема представительности, поставленная в разд. 5.2.2, совпадают в случае естественной рабочей нагрузки. Это легко понять, если предположить, что нас интересует только один индекс производительности — среднее случайной переменной, например времени ответа. Наше определение представительности требует, чтобы значение этого среднего, вычисленное в случае, когда система управляется выбранной естественной моделью рабочей нагрузки, совпадало со значением среднего, вычисленным при реальной рабочей нагрузке. Поскольку мы имеем дело со случайными переменными и статистиками, оценивающими такие описатели распределений, как среднее, это равенство должно быть определено статистически в терминах уровня значимости или доверительного интервала. Таким образом, для того чтобы получить представительную естественную модель, достаточно решить задачу сходимости по вероятности для данного доверительного интервала. Когда имеются два или более индекса производительности, все упомянутые выше типы моделей рабочих нагрузок являются представительными, только если критерий сходимости по вероятности удовлетворяется для всех индексов.

Если могут быть сделаны обоснованные предположения о стационарности, то способность естественных моделей точно представлять реальную рабочую нагрузку потенциально является очень высокой. Однако только потенциально, из-за дополнительного условия о том, что должна быть правильно решена задача о сходимости по вероятности. Системная независимость таких моделей не очень велика. Не существует общей гарантии, что критерий сходимости по вероятности будет удовлетворяться после даже маленькой модификации системы (заметим, что исследователя обычно интересуют изменения, влияющие на системные индексы производительности). Поэтому рекомендуется повторное вычисление дисперсии и доверительного интервала после любой модификации системы.

То, что естественные модели так мало управляемы (исследователь задает только времена и длительности измерений), делает их весьма негибкими и плохо воспроизводимыми. Некоторую степень опосредованной гибкости можно обеспечить, исключив из вычислений индексов производительности некоторые сборные данные.

Нельзя, конечно, пренебрегать ценой построения естественных моделей (см. методы, описанные в разд. 2.6.4), но она ниже, чем цена других типов рабочих нагрузок. Иначе обстоит дело с ценой использования. Естественные модели значительно менее компактны, чем искусственные, поскольку они требуют более длинных серий измерений, а также сбора, хранения и сжатия большего объема данных.

Таким образом, естественным моделям следует отдавать предпочтение в тех случаях, когда самой важной целью является получение высокой степени представительности при небольшой цене построения, если допускается низкий уровень управляемости и высокая цена использования. Преимущество естественных моделей, из-за которого их предпочитают в ряде установок и исследований, в том, что система не должна целиком предоставляться под измерительные эксперименты.

5.3.3. Неисполняемые искусственные модели

В разд. 5.3.1 модели рабочих нагрузок, не представляющие собой подлинную выборку из производственной рабочей нагрузки, классифицированы как *искусственные*. Мы также решили рассматривать *исполняемые* модели (те, что могут управлять реальной вычислительной системой) отдельно от *неисполняемых*. Очевидно, что и исполняемые искусственные модели, и естественные модели годятся для измерительных исследований (и для некоторых специальных типов программно-управляемых модельных экспериментов), в то время как аналитические и имитационные исследования используют неисполняемые искусственные модели.

Спектр неисполняемых моделей очень широк. Этот класс моделей включает, например, описание в терминах двух детерминированных переменных (время обработки записи центральным процессором и длина записи) для аналитической модели, описанной в разд. 4.2.2, а также очень детализированные трассировочные записи, используемые для управления трассирующим имитатором (см. разд. 3.2.4). В этом разделе мы не будем пытаться ни классифицировать формально неисполняемые модели, ни исследовать все многообразие входящих в этот класс моделей. Некоторые такие модели описаны в гл. 3 и 4, некоторые обсуждаются в этой главе (см. 5.1 и 5.2), а другие вводятся в следующих главах. Исследование того, как реальная рабочая нагрузка описывается в аналитических и имитационных моделях, приведенных в этой книге (по образцу исследования, проведенного в примерах 5.1 и 5.2), обнаружение аналогий и разработка новых моделей рабочих нагрузок с помощью экстраполяции — все это оставляется читателю в качестве полезных упражнений. Далее мы обсудим только построение некоторых типов неисполняемых моделей и их наиболее важные свойства.

В аналитических исследованиях вычислительных систем ограниченность математической трактовки часто диктует формулировку модели рабочей нагрузки, так же как и модели системы. Если модель системы сформулирована, то для исследователя остается очень мало возможностей. Эти возможности ограничены также требованием разрешимости. В моделях с очередями, например, распределения времен обслуживания в центрах обслуживания могут быть определены как часть модели рабочей нагрузки. Но использование измеренных распределений или желание учесть корреляции и автокорреляции времен обслуживания в общем случае ведет к неразрешимости модели в замкнутой форме. Исследователь обычно понимает эти проблемы, когда принимает решение применить аналитический метод и формулирует модель, зная об ограниченности такого подхода.

Когда расширяются границы множества разрешимых моделей с очередями, ограничения на описание рабочей нагрузки становятся более слабыми.

Отмеченный в гл. 4 факт, что сети с очередями с различными клас-

сами пользователей могут быть решены при определенных предположениях, позволяет моделировать поведение реальных заданий значительно точнее, чем в случае, когда можно оперировать только одним классом. Однако представительность даже очень усложненных моделей рабочих нагрузок, используемых в аналитических моделях, остается все еще довольно слабой. Получение представительной модели рабочей нагрузки никогда не бывает главной целью при ее формулировке, в которой обычно доминируют математические результаты. Разумеется, исследователю не следует забывать и о проблеме представительности. Как при формулировке модели рабочей нагрузки, так и при ее построении, которое состоит только в присваивании числовых значений параметрам рабочей нагрузки (например, специфицируется, что времена обслуживания в центре обслуживания i распределены экспоненциально со средним $1/\mu_i$), точность модели должна быть максимизирована в пределах ограничений, налагаемых математической разрешимостью. В разд. 4.4 рассматривается оценка модельных параметров по измеренным данным, которая необходима для построения модели рабочей нагрузки.

Аналитические модели рабочей нагрузки очень гибки применительно к изменениям тех аспектов реальной рабочей нагрузки, которые они моделируют. Легко представить изменения в среднем времени обслуживания, запрашиваемом заданиями в центре обслуживания сети с очередями, между тем как учесть изменения в запросах заданиями объема памяти может быть очень сложно или вовсе невозможно. К тому же такие модели рабочих нагрузок несложны в построении, очень компактны и отлично воспроизводимы. Системная зависимость моделей, прокомментированная на частной модели из примера 5.3, несущественно отличает их от других моделей рабочей нагрузки, основанных на сходных описаниях. Так как их представительность обычно невелика, то это не может очень повредить способности модели отражать изменения системы.

Модели рабочей нагрузки, используемые в имитационных моделях, управляемых распределениями, имеют более или менее те же свойства, что присущи моделям с очередями. Главное отличие составляет значительно больший потенциал представительности из-за снятия большинства ограничений на разрешимость. Так, всегда приемлемы произвольные распределения параметров, а корреляции и автокорреляции между параметрами (хотя они редко существуют) могут приниматься в рассмотрение с привлечением более сложной стохастической модели каждого задания, чем та, что описана в разд. 3.2.4. Можно легко моделировать запросы на ресурсы, которые могут совместно использоваться несколькими заданиями, такие, как объемы основной и вспомогательной памяти, разделяемые программы и базы данных. Легко также представить в имитационной модели одновременное использование более чем одного ресурса (например, центрального процессора и канала ввода-вывода). Эти модели, подобно аналитическим, очень легко управляемы, а следовательно, гибки и

воспроизводимы. На самом деле бóльшая степень детализации и отсутствие жестких ограничений делают их более гибкими, чем их аналитические аналоги. Они могут адекватно отражать значительно больше типов изменений реальной рабочей нагрузки, включая изменения в распределениях параметров рабочей нагрузки и в поведении классов заданий. Они так же просты в построении, как и аналитические модели, но требуют, как правило, больше информации, что объясняется большей степенью детализации и значительно большей ценой использования из-за того, что эти модели управляют имитационной моделью.

Как было подчеркнуто в разд. 3.2.4, в котором обсуждаются модели рабочей нагрузки для имитационных моделей, свойства трассировочных записей и программных представлений довольно существенно отличаются от свойств таких моделей для управляемых распределениями моделей. Трассировочные записи среди неисполняемых моделей ближе всего по характеру и свойствам к исполняемым моделям. Их представительность потенциально значительно выше, чем у рассматриваемых до сих пор моделей. Наказание за превосходную представительность трассировочных записей может выразиться в их громоздкости. Тем не менее, поскольку цена прогона управляемой трассировочной записью модели в общем случае не выше цены прогона модели любого другого типа, это наказание ограничивается дополнительным объемом памяти, необходимой для трассировочной записи, и потребностью доступа к ней в течение моделирования. Большая часть расходов по построению приходится на измерительные эксперименты для трассировки системы (эта процедура часто бывает довольно дорогой).

Трассировочные записи хорошо воспроизводятся, но, к сожалению, не обладают гибкостью. Этот недостаток делает очень сложным или даже невозможным приспособление модели к некоторым изменениям в системе, применительно к которым трассировочная запись оказывается непригодной. Поэтому важно определить типы изменений системы, которые могут понадобиться при моделировании, до формулировки модели и разработки эксперимента по трассировке. Это решение и степень, в которой оно учитывается, определяют область пригодности трассировочной записи, причем расширить эту область нелегко. Часто встречающийся разумный подход состоит в создании очень подробной трассировочной записи и сокращении ее перед использованием до более управляемых размеров. Этот подход позволяет исследователю уменьшить степень детализации модели рабочей нагрузки и, следовательно, во время моделирования иметь возможность расширить область пригодности просто повторной обработкой собранных данных без повторной трассировки системы. Обсудим теперь кратко пример недостатка гибкости трассировочной записи.

Пример 5.4. Рассмотрим трассировочную запись, изображенную на рис. 3.14 (а) и воспроизведенную для удобства на рис. 5.5(а). Трассировочная запись является моделью единственного задания и представляет последовательные запросы задания на ресурсы ПРОП-системы, описанной в примере 3.1.

Как нужно изменить модель, если мы хотим использовать ее для представления задания, у которого часть вычислений может перекрываться с некой работой по вводу-выводу? Ясно, что строго последовательная трассировочная запись на рис. 5.5(а) потребовала бы неких нетривиальных структурных изменений.

Возможное решение — добавить в каждом элементе «ввод-вывод» трассировочной записи третье поле, как показано на рис. 5.5(b). Это поле содержало бы

ЦП	55
Диск	120
ЦП	272
Диск	63
ЦП	137
Барабан	16
ЦП	84

–
141
–
525
–
0
–

Диск	55	120	196
Диск	327	63	852
Барабан	464	16	464

ЦП	25
Стоп	–

(a)

–
–

(b)

Стоп	–	–	–

(c)

Рис. 5.5. Изменения, которые требуется внести в исходную трассировочную запись задания (а), чтобы представить задание с частичным перекрытием вычислений и действий по вводу-выводу.

указание того процессорного времени (относительно начала процессорного интервала в следующем элементе копии), когда операция по вводу-выводу должна закончиться, что позволит центральному процессору продолжить выполнение задания. Например, запись 141 на рис. 5.5(b) означает, что первый запрос диска разрешает центральному процессору выполнять задание в течение 141 единицы времени, начиная с момента выдачи запроса. Если за это время запрос все еще не был удовлетворен, то центральный процессор останавливается и ждет завершения работы с диском.

Лучшая модель — трассировочная запись — для такого задания изображена на рис. 5.5(c). Записи во втором поле — это чистые процессорные времена (измеренные от начала выполнения), в которые инициируются операции ввода-вывода, причем тип операции закодирован в первом поле. Четвертое поле содержит для каждой операции ввода-вывода время, в которое операция должна быть совершенно закончена, с тем, чтобы можно было позволить продолжить вычислительную работу задания. Трассировочная запись на рис. 5.5(c) представляет то же задание, что и на рис. 5.5(a) и 5.5(b). Очевидно, что добавление к трассировочной записи на рис. 5.5(a) дополнительного поля [рис. 5.5(b)] или преобразование его в трассировочную запись с рис. 5.5(c) требует нетривиальных усилий. Если первоначальное задание моделируется тройкой ($t_{цп}$, $t_{бр}$, $t_{дс}$) случайных величин, то разрешение возможности перекрытия повлекло бы только добавление одной (или более) новой случайной величины, которая будет представлять предельные значения, показанные на рис. 5.5(b) или в четвертом поле рис. 5.5(c).

Таблица 5.4

Свойства неисполняемых искусственных моделей

	Модели рабочей нагрузки, представленные			
	аналитическими методами	моделью, управляемой распределением	программно-управляемой моделью	моделью, управляемой трассировочной записью
Возможности:				
Представительность	Плохая	Приемлемая	Хорошая	Отличная
Компактность	Отличная	Очень хорошая	Приемлемая	Плохая
Гибкость	Хорошая	Хорошая	Приемлемая	Плохая
Воспроизводимость	Отличная	Отличная	Отличная	Отличная
Системная независимость	Плохая	Приемлемая	Хорошая	Хорошая
Цена:				
Построения	Низкая	Низкая — средняя	Средняя — высокая	Высокая
Использования	Низкая	Высокая	Высокая	Высокая

Таблица 5.4 подводит итоги качественного обсуждения свойств неисполняемых моделей рабочей нагрузки. Таблица показывает сравнительные возможности различных типов моделей в терминах характеристик, введенных в разд. 5.1.

5.3.4. Исполняемые искусственные модели

Любая модель рабочей нагрузки, состоящая из одной или более программ, исполняемых на оцениваемой системе, но не удовлетворяющих определению естественной модели (см. разд. 5.3.2), называется исполняемой искусственной моделью. Исполняемые модели используются почти исключительно в измерительных экспериментах. Единственным исключением являются программно управляемые имитационные модели, получающие на вход реальные программы. Основные причины, по которым искусственные модели рабочей нагрузки предпочитаются естественным во многих измерительных экспериментах, состоят в их лучшей воспроизводимости, бóльшей гибкости и потенциально бóльшей компактности. Как было показано в разд. 5.3.2, имеются области, в которых естественные модели особенно плохи. С другой стороны, искусственные модели, как правило, менее представительны и более системно зависимы, чем их естественные аналоги. Они в общем более громоздки в построении и требуют всей системы на время серии измерений, что очень неудобно для многих установок.

Исследователь, пользующийся измерительными методами, имеет возможность выбрать наиболее подходящую для данного эксперимента модель рабочей нагрузки, соответствующую целям исследования и объему исследования, а также специфическим проблемам и характеристикам данной установки. Если целью эксперимента является, скажем, измерение влияния производительности некой не полностью отлаженной модификации системы на операционную систему, то искусственная модель рабочей нагрузки, управляющая измененной системой в течение периодов, зарезервированных для нужд персонала установки, будет устранять неудобства, которые могут возникнуть у пользователя неотлаженной системы, и позволит лучше воспроизвести среду системы, чем в естественной модели. Последнее свойство особенно желательно в сравнительных исследованиях. С другой стороны, если установка постоянно доступна коллективу пользователей, то естественным моделям альтернативы нет. В этом случае приемлемый уровень воспроизводимости можно обеспечить только в ущерб компактности (т. е. сделав серии измерений достаточно длинными), а пользователи будут подвергаться риску работы с неотлаженной операционной системой.

Экспериментом, заведомо требующим искусственной модели, является попытка испытать поведение системы при реальной рабочей нагрузке, совпадающей не с производственной рабочей нагрузкой, а скорее с предполагаемой в будущем нагрузкой. Напротив, эксплуатационные исследования, целью которых являются ответы на вопросы, касающиеся производительности системы при производственной рабочей нагрузке (например, является ли последнее изменение системы полезным?), в общем случае требуют естественной модели. Это, конечно, справедливо, если не была построена искусственная модель рабочей нагрузки. На самом деле данные, нужные для построения, можно собрать только при управлении системой с помощью естественной модели.

Наше сравнение исполняемых искусственных моделей и естественных моделей не должно привести пользователя к ошибочному выводу, что все исполняемые искусственные модели имеют одни и те же характеристики. В этом и в следующих разделах мы увидим, что существует большое разнообразие моделей, имеющих существенно разные свойства. Таким образом, исследователь, решивший, согласно намеченному выше критерию, использовать искусственную модель для измерительного эксперимента, все еще имеет возможность определить некоторые факторы перед и во время построения такой модели.

В разд. 5.2.1 проводилось различие между описанием рабочей нагрузки, представляющим ее целиком, и таким, которое состоит из совокупности описаний заданий. Одной из первых искусственных моделей, предложенных для оценки производительности, была *смесь команд* [см. Рейчелсон и Коллинз (1964) и Арбакл (1966)], попадающая в последнюю категорию. Модель смеси команд для реальной рабочей нагрузки представляет собой частотное распределение типов

команд, выполняемых во время обработки рабочей нагрузки. Набор этих частот представляет относительную используемость отдельных машинных команд для данной рабочей нагрузки, или, другими словами, их относительные запросы на процессорные ресурсы.

Смеси команд, конечно, невыполнимы. Мы описываем их в этом разделе потому, что из них прямо выводятся исполняемые модели. Любая программа, характеризующаяся данным набором частот команд, является такой моделью. Заметим, что представленная так рабочая нагрузка может состоять из единственного задания. Таким образом, смесь команд может использоваться и в качестве модели задания. Если машинные команды разбиты на группы (согласно, например, их временам выполнения), то смесь команд может состоять из частот групп. Организация аппаратуры, измеряющей эти частоты, описана в разд. 2.3.3 (см. рис. 2.6).

Выполняемая смесь команд может использоваться для измерения общего времени центрального процессора, нужного заданию или рабочей нагрузке. Неисполняемые смеси команд позволяют нам вычислить те же запросы, если известно время выполнения t_{xi} команды i для всех i; относительная частота команды i обозначается f_i. *Среднее время выполнения команды* будет

$$\bar{t}_x = \sum_i f_i t_{xi} \quad \left(\sum_i f_i = 1\right). \tag{5.9}$$

Общее время центрального процессора, нужное для выполнения n команд:

$$t_x(n) \approx n\bar{t}_x, \tag{5.10}$$

а *средняя интенсивность выполнения команд ЦП* для данной рабочей нагрузки (учитывая только вычислительные запросы заданий) будет $v_{\text{цп}} = 1/\bar{t}_x$. Когда измерение реальной рабочей нагрузки слишком трудоемко или вовсе невозможно (потому, например, что системы просто не существует или она еще не смонтирована), исследователь может воспользоваться смесью команд, измеренной на аналогичной установке. В этом случае может оказаться полезной *стандартная смесь команд*. Наиболее известным опубликованным примером стандартной смеси является *смесь Гибсона*, одна из сокращенных версий которой приведена в табл. 5.5. Гибсон (1970) получил частоты в этой смеси, исходя из большого объема данных, собранных на установках IBM 7090. Таким образом, эта смесь отражает общую относительную используемость команд системы 7090 для научных и технических приложений.

Смесь команд очевидным образом зависит от архитектуры центрального процессора. Те же потребности в обработке информации, выраженные на языке другой машины, дадут другую смесь команд, причем отличие будет не только в частотах, но и в типах команд, присутствующих в смеси. Можно уменьшить зависимость от системы (как указано в разд. 5.2.3), выбирая такие параметры рабочей нагрузки, которые представляют скорее логические, чем физические ресурсы. Для случая смеси это можно сделать, группируя инструкции в более широкие классы, имеющиеся в большинстве центральных процессоров. Приведенная в табл. 5.5 форма представления смеси Гибсона является примером классификации такого типа. Мы видим,

Таблица 5.5

Две стандартные смеси команд

Класс команд	Смесь Гибсона f_i [%]	Смесь Флинна f_i [%]
Загрузка/запоминание	31.2	45.1
Индексная арифметика	18	
Ветвление	16.6	27.5
Сравнение	3.8	10.8
С фиксированной точкой	6.9	7.6
С плавающей точкой	12.2	3.2
Сдвиг/логические	6.0	4.5
Другие	5.3	1.3
	100.0	100.0

Источник: Флинн (1974).

что такая форма допускает прямое сравнение с другой смесью, также приведенной в табл. 5.5. Эта смесь получена для научно-технических программ, выполнявшихся на установках системы IBM/360, и опубликована Флинном (1974).

Смесь, выраженная в обоснованно системно-независимой форме, позволит нам (как это будет показано в следующем примере) сравнить две или более архитектур центрального процессора и оценить их удобство для установки, порождающей эту смесь. Две смеси, такие, как в табл. 5.5, которые основываются на данных, собранных на большом числе установок, не могут представлять отдельную установку. Поэтому они могут использоваться, как это и было сделано Флинном, для сравнения свойств разных архитектур и для анализа возможных временных отклонений от типичного использования.

Пример 5.5. Два процессора A и B — главные кандидаты для новой научной установки. Поскольку вырабатываемая коллективом пользователей рабочая нагрузка неизвестна и не может быть измерена до тех пор, пока система не будет собрана, то мы будем пользоваться смесью Флинна как ее приблизительной моделью (наборы команд процессоров A и B не слишком отличаются от набора для системы 360). Каждая команда этих двух машин должна быть отнесена к одному из классов из табл. 5.5, а их относительные частоты должны быть оценены. Из этих частот и времен выполнения команд, приведенных в руководствах к этим двум процессорам (которым можно верить только потому, что в данном исследовании сделаны очень грубые допущения), мы можем вывести средние взвешенные времена выполнения для команд каждого класса. Пусть это будут те времена, которые приведены в табл. 5.6. Тогда, используя

(5.9), получаем

$$\bar{t}_{xA}=1.88 \text{ мкс}, \quad \bar{t}_{xB}=1.67 \text{ мкс} \tag{5.11}$$

Поэтому при сделанных допущениях при обработке смеси Флинна мы можем прийти к выводу, что машина А примерно на 11% медленнее машины В.

Таблица 5.6

Средние взвешенные времена выполнения классов команд для двух сравниваемых центральных процессоров (пример 5.4)

Класс команд	Машина А t_{x_i} [мкс]	Машина В t_{x_i} [мкс]
Загрузка/запоминание Индексная арифметика	1.8	1.3
Ветвление	1.2	1.3
Сравнение	2.4	2.0
С фиксированной точкой	2.4	2.6
С плавающей точкой	5.4	5.6
Сдвиг/логические	1.8	2.0
Другие	3.5	4.0

Если повторить эти вычисления, но использовать при этом смесь Гибсона из табл. 5.5, то мы получим другие абсолютные результаты ($\bar{t}_{xA}=2.29$ мкс, $\bar{t}_{xB}=2.12$ мкс), но аналогичные выводы. А именно, при смеси Гибсона машина А будет приблизительно на 7% медленнее машины В. С другой стороны, смесь, состоящая из ветвлений и логических операций должна обрабатываться быстрее на машине А, чем на машине В. Некоторые дополнительные примеры можно найти у Драммонда (1973). ■

Могут оказаться очень полезными смеси, представляющие нагрузки, более специализированные, чем просто научные или экономические (или нагрузки для приложений в разных областях). Например, обе смеси в табл. 5.5 содержат данные о компиляции, равно как и об выполнении научных программ. Если отделить компиляционную смесь от исполнительной, то можно изучать свойства архитектуры центрального процессора относительно этих двух типов обработки данных [см. Флинн (1974)] и моделировать рабочие нагрузки, ориентированные на компиляцию или на выполнение.

Смеси могут плодотворно использоваться и в языках более высокого уровня, чем машинный. Для языков уровня ФОРТРАНа, АЛГОЛа или АПЛ мы имеем *смеси предложений*. На уровне языка управления заданиями или языка диалога мы имеем *смесь управляющих предложений* или *команд*. В табл. 5.7 приводится пример смеси предложений, собранной Кнутом (1971) на основании 24 программ на ФОРТРАНе. Отметим сделанное в таблице важное различие между статическими частотами, полученными из листинга программ, и динамическими, измеренными в процессе выполнения. Динамические

Таблица 5.7

Смесь предложений ФОРТРАНа

Тип выполняемого предложения	Относительная частота f_i [%]	
	статическая	динамическая
Присваивание	51	67
IF*	10	11
GOTO	9	9
DO	9	3
CALL	5	3
WRITE	5	1
CONTINUE	4	7
RETURN	4	3
READ	2	0
STOP	1	0

Источник: Кнут (1971).

* Предложение IF (*a*)*x* засчитывалось как IF и как предложение типа *x*. Поэтому сумма частот может быть больше 100 (случай динамических частот в данной таблице).

смеси представляют для нас больший интерес, так как их можно использовать в качестве моделей рабочей нагрузки. Специальный тип смеси предложений, использованный Морганом и Кэмпбеллом (1973) для моделирования частных программ, рассматривается в разд. 6.3.4. В табл. 5.8 приводится смесь команд, измеренная Шерром (1967) на установке CTSS, в которой все команды системы разбиты на пять категорий.

Как и все модели, основанные на простом распределении, смеси не содержат информации о возможных корреляциях и других взаимо-

Таблица 5.8

Смесь диалоговых команд

Класс команд	Относительная частота
Обработка файлов	36
Ввод-вывод и редактирование	15
Прогонка и отладка программ	12
Ассемблирование и компиляция	9
Прочие	28

Источник: Шерр (1967).

действиях, которые могут иметь место в моделируемом явлении. Неявное предположение, обосновывающее пользу смесей, состоит в том, что на поведение системы оказывают влияние только отдельные выполняемые команды (или предложения), а не их последовательность. Когда исследуется система — неконвейерный центральный процессор, имеющий свою собственную память, и нас интересует только процессорное время, необходимое для обработки определенной рабочей нагрузки, тогда такое допущение будет удовлетворительным. Однако с помощью смеси команд нельзя представить взаимодействия между выполнениями последовательных команд, которые сильно влияют на производительность конвейерной машины. Нельзя представить и взаимодействия между центральным процессором и другими аппаратными ресурсами системы. Иначе говоря, посредством использования смеси команд нельзя определить действительное время обработки, т. е. пропускную способность системы. Описанные выше другие типы смесей имеют такие же характеристики. Они могут использоваться для оценки логических машин в представлении пользователей (например, ФОРТРАН-машины или машины диалогового языка), но только при отсутствии взаимодействий как между предложениями (или командами), так и между другими процессами, работающими в той же системе, но не представленными в модели рабочей нагрузки.

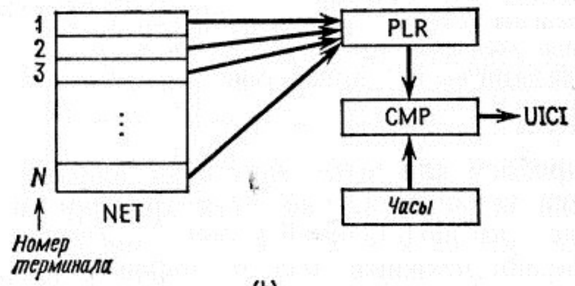
Другой тип исполняемой искусственной модели, который нельзя создать, просто собрав вместе отдельные модели заданий,— это *исполняемая трассировочная запись*. Эта модель получается трассировкой производственной рабочей нагрузки системы и прокручиванием полученной трассировочной записи каждый раз, когда нужно провести измерительный эксперимент. В случае пакетной обработки трассировочная запись будет состоять из хронологически упорядоченных колод заданий из рабочей нагрузки. В диалоговой установке в трассировочную запись должны попасть команды пользователя и длительность терминальной части для каждого взаимодействия. Исполняемую трассировочную запись можно также получить из вероятностной модели рабочей нагрузки. Однако, как и в гл. 3, мы будем называть «трассировочной записью» только то, что получено из прямых измерений. Использование вероятностных моделей для управления системой обсуждается в разд. 5.3.5.

Пример 5.6. Исполняемая трассировочная запись диалоговой системы и возможный способ ее использования для моделирования N активных терминалов схематически изображены на рис. 5.6. В данном решении управление системой состоит из двух программ обработки прерываний CPC и UIC (их блок-схемы изображены на рис. 5.6(b)), таблицы следующего события (NET на рис. 5.6 (b)) и буферной области, в которую различные разделы трассировочной записи последовательно считываются из вспомогательной памяти. Предполагается, что система снабжена часами с программно-загружаемым регистром PLR и блоком сравнения CMP (см. разд. 2.3.2).

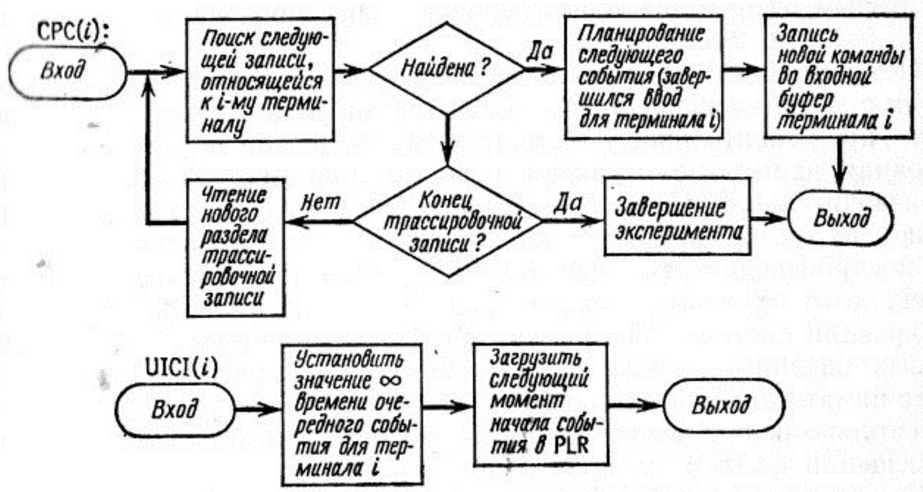
Для данного управления работой системы требуются два типа сигналов прерывания: Прерывание по завершению ввода пользователя (UICI) и Прерывание по завершению обработки команды (CPCI). Моменты, в которые происходят эти прерывания, определены на рис. 1.4.

CPCI генерируется завершением процесса, как обычно при работе системы. Небольшое изменение подпрограммы операционной системы, обычно обрабатывающей завершение процесса, вызывает подпрограмму CPC, когда завершается выполнение команды. Как показано на рис. 5.6 (c), подпрограмма CPC обновляет массив NET, который содержит указание времени очередного события для каждого терминала.

(a)



(b)



(c)

Рис. 5.6. Схема управления диалоговой системы с помощью исполняемой трассировочной записи (см. пример 5.6.): (a) — формат трассировочной записи; (b) — планирование очередного события и моделирование терминальной части взаимодействия, (c) — блок-схемы подпрограмм обработки прерываний.

Отметим, что событие представляет собой завершение ввода команды. При обновлении массива NET CPC использует информацию из соответствующей трассировочной записи [см. рис. 5.6 (a)]. Потом CPC записывает вводимую команду в запланированный момент времени на буфер терминального ввода и передает управление операционной системе. Вывод, происходящий во время процесса счета, следует на самом деле посылать на терминалы, чтобы сделать эксперимент более правдоподобным. Когда вывод

с терминалов оказывается слишком неудобным, можно внести в операционную систему изменения, чтобы воспрепятствовать отправлению выходных сообщений на терминалы, и можно вызвать пустой процесс, потребляющий те ресурсы, которые обычно потребляются на управление терминальным вводом-выводом.

Другой сигнал прерывания, UICI, генерируется блоком сравнения CMP, когда время на часах совпадает со временем очередного события. Подпрограмма UIC вызывается и загружает в PLR наименьший элемент массива NET, запомнив при этом соответствующий номер терминала. Отметим, что UIC определяет наименьший элемент в NET, исключив из рассмотрения терминал, который вызвал прерывание. Наконец, UIC передает управление операционной системе, возобновляющей процесс принятия команд.

Читатель, конечно, заметил аналогичность методов, описанных в данном примере, и методов, описанных в гл. 3,— и те, и другие моделируются. В частности, метод очередного события, введенный в примере 3.2, был применен здесь для моделирования одновременной активности N терминалов. ■

Одной из проблем для исполняемых трассировочных записей является проблема искажений, вносимых трассирующим измерителем при трассировке производственной рабочей нагрузки. Эти искажения могут существенно исказить модель рабочей нагрузки, особенно записываемые времена событий. Другая проблема — выбор длины трассировочной записи. Выбор должен осуществляться согласно тем же критериям, что и для естественных моделей рабочей нагрузки, к которым очень близки исполняемые трассировочные записи (см. разд. 5.3.2 и 2.6.4).

Потенциальная представительность трассировочной записи в сравнении с другими искусственными моделями очень высока, но немного ниже представительности естественных моделей, поскольку трассировочная запись не отражает реакции пользователя на изменения производительности. Как и естественные модели, трассировочные записи не очень гибки, но, как и у других искусственных моделей, их воспроизводимость значительно лучше, чем у естественных моделей, хотя их использование каждый раз требует тщательной инициализации системы. Например, все файлы, используемые в записываемых заданиях, также должны быть перезаписаны, причем в те места памяти, в которых они были расположены во время трассировки, если только целью исследования не является как раз оценка влияния размещения файлов на производительность.

Если описание рабочей нагрузки, на которой основывается трассировочная запись, не дает представительности с достаточной степенью системной независимости, то может оказаться очень трудно или вовсе невозможно использовать эту трассировочную запись из-за малой гибкости, даваемой такими моделями (см. пример 5.4). Компактность, конечно, не является свойством исполняемых моделей, но цена их разработки не чрезмерна.

Трассировочные записи использовались в некоторых измерительных исследованиях. Например, Сакман (1964) измерил производительность системы SAGE для экспериментов по *возобновляемым* (или *регенеративным*) записям, заключающимся в управлении системой

с помощью исполняемых трассировочных записей [см. также Каруш (1969)]. Упоминаемые выше очень сложные технологические проблемы (в частности, такие, как искажения при измерении и инициализации) служат объяснением того, почему возобновляемые эксперименты до сих пор мало распространены.

Рис. 5.7. Фаза построения исполняемой искусственной модели, составленной из отдельных заданий.

Для того чтобы получить исполняемую модель объединением моделей отдельных заданий, мы должны в принципе произвести следующие две операции (см. рис. 5.7).

A. Определить значения параметров задания для каждого из заданий, которые должны появиться в модели. Мы предполагаем, что параметры задания отбираются при формулировке модели. Как отмечено в разд. 5.1, действительность часто значительно сложнее идеализированных предположений.

B. Построить модель каждого задания, охарактеризованного согласно п. A.

Различные подходы к этим двум проблемам будут рассмотрены в разд. 5.3.5. В конце этого раздела мы коротко обсудим исполняемые модели рабочей нагрузки, которые строятся перестановкой шагов (т. е. шаг B выполняется перед шагом A). Существовавшие ранее или специально построенные модели заданий отбираются и объединяются вместе (возможно, с присвоением каждой из них некоего веса) согласно априорным и часто очень поверхностным знаниям о том, какие типы работ будут выполняться на установке. Целью такого подхода является получение не слишком грубой картины ожидаемой нагрузки установки или в некоторых случаях получение ввода, подходящего для проверки определенных частей или свойств системы [см., например, Каруш (1969)]. Такие стандартные модели заданий, предназначенные для выбора вычислительной машины, называются *ядрами* [см., например, Арбакл (1966) и Лукас (1971b)] или, иногда, *стандартизированными эталонами* [см. Хиллигасс (1966)]. Для этой цели были предложены *стандартизированные синтетические модули* [см. Лукас (1971a)]. Определения эталонов и синтетических заданий будут даны в разд. 5.3.5.

Модели рабочей нагрузки, подходящие для диалоговой системы, могут рассматриваться как принадлежащие к категории ядер, составляемых из сценариев. *Сценарий* представляет собой последовательность диалоговых команд, разделенных (обычно фиксированными)

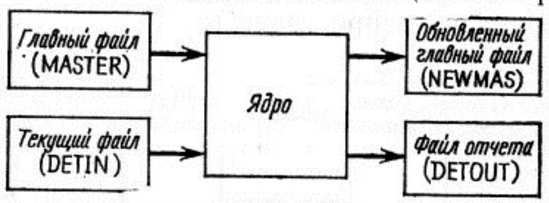


Рис. 5.8. Ядро обновления файла.

интервалами для обдумывания. Предполагается, что эта последовательность представляет собой типичный сеанс работы пользователя за терминалом. Таким образом, набор сценариев может использоваться для моделирования группы терминальных пользователей, одновременно работающих с диалоговой системой.

Типичные примеры задач-ядер для научных установок — обращение матрицы, приближенное вычисление квадратного корня, вычисление значения полинома; для экономических установок — ядро обновления файла. В последнем случае главный файл читается с диска или ленты и обновляется исходя из содержимого текущего файла и генерируется файл отчета, содержащий записи о произведенных пересылках (рис. 5.8). Пример сценария для диалоговой научной установки — сценарий Multics, описанной Зальцером и Джинтеллом (1970) и обобщенный в табл. 5.9.

Таблица 5.9

Последовательность операций в диалоговом сценарии

Операция	Комментарии
Регистрация пользователя	—
Создание программы	Отредактированы некоторые ошибки печати; сделаны некоторые синтаксические ошибки
Компиляция программы	Диагносцированы синтаксические ошибки
Отладка программы	Синтаксические ошибки удалены
Компиляция программы	Компиляция успешно завершена
Выполнение программы	Выполнение успешно завершено, результаты выданы на печать
Распечатка программы	Текст программы выдан на печать
Распечатка файлов	Список файлов, связанных с программой, выдан на печать
Удаление программы	—
Регистрация выхода пользователя из системы	—

Источник: Зальцер и Джинтелл (1970).

Использование множества ядер вполне оправдано при планировании новой установки, когда о рабочей нагрузке, которую будет обрабатывать система, можно только догадываться, и для сравнения характеристик разных машин нужна системно-независимая модель рабочей нагрузки. В последнем случае в процессе выбора вычислительной системы пользователи иногда включают задачи-ядра в свои спецификации. Будущие поставщики должны закодировать эти ядра для своих предполагаемых машин и измерить или оценить времена счета. Даже если наборы ядер, стандартизованных эталонов или сценариев довольно хорошо отражают мысли исследователя о рабочей нагрузке установки, иногда они очень отдаленно напоминают действительную рабочую нагрузку. Их область применения определенно шире, чем у смесей команд. Оценка конвейерного компьютера и моделирование операций ввода-вывода — два примера задач, в которых применение ядер дает значительно бóльшую точность, чем смеси. Другое важное свойство таких моделей рабочих нагрузок — потенциально высокая системная независимость ядер.

5.3.5. Построение исполняемой искусственной модели

В предыдущем разделе задача построения исполняемой модели рабочей нагрузки с помощью объединения моделей отдельных заданий была разделена на два концептуально разных шага, A и B, изображенных на рис. 5.7.

Шаг A заключается в определении параметров каждого из заданий, которое должно быть включено в модель. Два противоречивых требования — представительности и компактности — играют важную роль при выборе этих заданий. Бóльшая модель рабочей нагрузки потенциально более представительна, но ее прогон занимает больше времени. Стоимость эксперимента с искусственной рабочей нагрузкой возрастает при увеличении его длительности, в особенности из-за того, что в течение этого эксперимента пользователь полностью занимает систему. Длительность серии зависит не только от числа заданий, но и от их длительности. Модель рабочей нагрузки потенциально более представительна и системно-независима, если характеристики ее заданий не слишком отличаются от характеристик большинства реальных заданий. Таким образом, большее число заданий нормальной длительности следует предпочитать меньшему числу заданий с длительностью, значительно превышающей обычную. Бóльшая потенциальная представительность и системная независимость происходят из-за бóльшей вероятности того, что будут получены похожие типы и интенсивности взаимодействий между компонентами системы и компонентами рабочей нагрузки.

На шаге A максимальное время обработки модели рабочей нагрузки может быть одной из проектных спецификаций. В тех экспериментах, в которых измеряется общее время выполнения для оценки средней пропускной способности системы, максимальное время вы-

полнения не может быть специфицировано заранее. Однако его порядок может быть (и часто бывает) включен в проектные требования. Выбор этого порядка устанавливает ограничение на степень достижимой моделью рабочей нагрузки представительности. В пределах этого ограничения разработчик должен сделать модель по возможности представительной.

Другие цели разработки могут подчеркивать некоторые свойства из перечисленных нами в табл. 5.1 (в пределах ограничений для данного типа моделей рабочей нагрузки, рассмотренных в начале разд. 5.3.4). Эти цели определят некоторые решения, которые нужно принять при построении модели. Рассмотрение различных альтернатив, которые появятся далее в этом разделе, поможет читателю понять, каким образом желательные свойства должны влиять на этот выбор.

Если предположить, что мы уже выбрали описание рабочей нагрузки и что это описание выражается в терминах v-строк параметров задания ($w_1, \ldots, w_v$), как это рассматривалось в разд. 5.2.1, то каким методом мы можем воспользоваться для определения значений параметров каждого из заданий, которое включается в модель? Пять возможных методов перечислены в табл. 5.10. Методы Aa и Ab применимы

Таблица 5.10

Методы определения параметров отдельных заданий (шаг A процедуры построения исполняемой модели)

Метод	Описание
Для параметров обоих типов:	
Aa	Построить вероятностные распределения параметров реальной рабочей нагрузки. С помощью выборки из этих распределений получить параметры каждого задания в модели
Ab	Сделать выборку реальных заданий из реальной рабочей нагрузки. Использовать параметры каждого задания для описания задания модели
Для скалярных параметров:	
Ac	Разбить задания реальной рабочей нагрузки на классы, охарактеризованные одинаковым набором параметров. Выбрать соответствующее число заданий из каждого класса и использовать параметры каждого задания для описания задания из модели
Ad	Построить совместное вероятностное распределение параметров для реальной рабочей нагрузки. Определить из этого распределения параметры множества заданий с таким же совместным распределением
Для векторных параметров:	
Ae	Определить состояния задания в терминах значений или диапазонов значений его параметров. Построить матрицы вероятностей переходов для реальной рабочей нагрузки и использовать их для определения последовательностей значений параметров каждого задания

для характеристик параметров любого типа, в то время как методы Ac и Ad удобны для скалярных величин, а метод Ae — для моделей векторного типа.

Самым несложным является *метод Aa*, поскольку в нем не делается попытки учесть корреляции и автокорреляции внутри и между параметрами задания. Распределения параметров задания $w_1, \ldots, w_v$ предполагаются известными или из измерений, или из оценок. v-строка значений для задания в модели рабочей нагрузки получается выборкой из этих распределений таким же образом, как и в простых моделях, управляемых распределением (см. разд. 3.2.4). Так как распределения w_i выбираются независимо, то в методе Aa неявно подразумевается, что на производительность системы не оказывают существенного влияния статистические отношения между w_i. Размер выборки, т. е. число моделей заданий, полученных из распределений, должен быть статистически значимым, с тем чтобы распределения параметров задания в модели можно было считать хорошими приближениями параметров реальной рабочей нагрузки. Это условие, к сожалению, в общем случае не является достаточной гарантией определенного уровня представительности для модели.

Этот метод был использован Шветманом и Брауном (1972), охарактеризовавшими производственную рабочую нагрузку установки CDC 6600 в Остинском университетском городке (Техасский университет) с помощью измеренных распределений таких параметров, как время между поступлениями заданий, заданные пользователем лимиты времени и памяти, разность между лимитом времени и использованным на самом деле временем, отношение между лимитом памяти и использованным на самом деле объемом памяти, отношение между использованным временем и временем центрального процессора, монопольная скорость передачи данных при вводе-выводе и монопольная скорость передачи данных для центрального процессора.

Метод Ab учитывает корреляции и автокорреляции, выводя v-строки из реальных заданий. Оправданность этого метода в том, что в реальных заданиях эти взаимосвязи присутствуют автоматически и их не нужно специально воспроизводить. Таким образом, вместо выборки из распределений параметров делается выборка из реальной рабочей нагрузки, которая в данном случае обычно совпадает с производственной рабочей нагрузкой. Могут использоваться различные стратегии выборки. Мы можем, например, выбирать каждое n-е из появляющихся заданий (n-е, $2n$-е и т. д.) или обрабатываемое в настоящий момент задание, или задание, выполняемое в момент периодической выборки (если такое задание существует). Стратегии, основанные на периодической выборке, вероятно, дают лучшую представительность моделей рабочей нагрузки, поскольку они придают больший вес тем заданиям, которые, больше используя ресурсы, сильнее влияют на производительность. Однако эти стратегии могут придавать чрезмерное значение «трудным» заданиям. Таким образом, для установок, чья рабочая нагрузка далека от однородной, нужно

использовать комбинации этих двух стратегий или другой способ.

Метод Ab — единственный метод, который позволяет нам в совокупности с методом Ba, изложенным ниже, избежать явного описания реальной рабочей нагрузки с помощью v-строк параметров, потому что выбранные задания могут использоваться прямо при построении модели рабочей нагрузки. Однако исследователь в малой степени распоряжается составлением такой модели, что может привести к меньшей, чем требуется, представительности, несмотря на большое число заданий. Это замечание станет для читателя убедительным, если он заметит сходство между моделью, полученной методом Ab, и исполняемой трассировочной записью. Последняя может рассматриваться как частный случай модели, полученной методом Ab. О приложениях этого метода сообщают Шоуп и др. (1970) и Вуд и Форман (1971).

Метод Ac дает моделировщику лучшие возможности в смысле представительности и, следовательно, компактности модели. Предположим, что параметры рабочей нагрузки таковы, что задание характеризуется одиночными значениями $w_1, \ldots, w_v$. Другими словами, эти параметры представляют скорее общие запросы на ресурсы, а не их историю. Тогда в *v-мерном пространстве параметров задания* каждое задание представляется точкой. Для классификации заданий мы можем применить метод разбиения совокупности на группы *(кластерный метод)* (он используется, например, в распознавании образов или в информационном поиске) для группировки точек, представляющих реальную рабочую нагрузку в пространстве параметров. Целью метода разбиения на группы является группировка моделей согласно некоторому критерию близости. Критерий может быть, например, такой: сумма квадратов расстояний между каждой точкой группы и центром тяжести группы должна быть минимальной. Алгоритмы разделения на группы часто требуют от их пользователя определения числа рассматриваемых групп. С ростом числа групп возрастает точность классификации и цена ее определения. Однако такой алгоритм может быть применен итеративно. Можно, например, начать с двух групп и применять алгоритм к возрастающему числу групп до тех пор, пока критерий точности не будет удовлетворен. Критерием точности может быть требование того, чтобы максимум сумм квадратов, характеризующих группу, был меньше заранее определенного числа.

В контексте метода Ac каждая группа рассматривается как класс заданий. Реальную рабочую нагрузку можно моделировать набором заданий, извлеченных из каждого класса пропорционально количеству его элементов, совершенно так же, как политическая система базируется на партиях и пропорциональном представительстве. Как и в политической системе, компактность требует, чтобы число представителей от каждого класса заданий было очень малой частью от числа элементов класса. Эти представители могут отбираться различными методами. Например, все v-строки q_i, представляющие класс i, должны быть одинаковы и представлять собой координаты центра тяжести класса i. Или они могут характеризовать те q_i реальных заданий, которые ближе всего к центру тяжести или которые удовлетворяют другим требованиям.

Результаты метода Ac бывают вполне приемлемыми, если группы хорошо определены и быстро ухудшаются при возрастании неопределенности их границ. Несмотря на то что формальные методы разбиения на группы не применялись, методы, аналогичные Ac, были использованы при построении некоторых моделей рабочих нагрузок. Например, Джослин (1965) качественно классифицирует реальные задания согласно их применению. Для установки с пакетной обработкой он разделяет рабочую нагрузку на следующие классы: написанные на ФОРТРАНе инженерные задачи, написанные на КОБОЛе коммерческие задачи, написанные на ФОРТРАНе математические задачи и т. д. Другие классификации, часто встречающиеся в литературе, берут за основу язык программирования, на котором написаны задания, или приоритеты, присвоенные заданиям программистами и системой. Хант и др. (1971) описывают рабочую нагрузку установки CDC 6400 в Вашингтонском университете с помощью параметров, перечисленных в табл. 5.11. Применение итеративного метода разбиения на группы, давшего в результате шесть классов заданий, описано в табл. 5.12. Первый шаг процедуры разбиения рабочей нагрузки на две группы, которые оказываются полностью совпадающими с исследовательскими и учебными задачами, приведен в табл. 5.11.

Таблица 5.11

Параметры задания для установки пакетной обработки

Параметр	Среднее значение		
	Общее	Исследовательские задания	Учебные задания
Прочитанные карты	224	430	95
Отпечатанные строки	760	1430	442
Время ЦП (с)	11.0	26	3.8
Время периферийного процессора (с)	11.9	22	7.1
Объем основной памяти (слов)	17.85K	21.12K	16.32K
Обращения к лентопротяжному устройству	0.28	0.4	0.22
Цена для пользователя ($)	1.44	3.40	0.48
Доля заданий, использующих ФОРТРАН	0.54	0.73	0.44
Число заданий	1588	527	1061

Источник: Хант и др. (1971), с разрешения AFIPS, INC.

Таблица 5.12

Классы заданий, полученных применением метода Ac к рабочей нагрузке из табл. 5.11

Класс	Численность	Преобладающий тип задания
1	408	Небольшие задания на ФОРТРАНе
2	472	Небольшие задания не на ФОРТРАНе (например, на BASIC)
3	86	Небольшие задания с аварийным окончанием
4	181	Средние задания, использующие ленты
5	293	Средние задания, не использующие ленты
6	148	Большие задания
	1588	

Источник: Хант и др. (1971).

Метод Ad концептуально аналогичен методу Ac и использует тот же тип описания, даже если не производятся попытки разбиения рабочей нагрузки. Уменьшенный образ этой рабочей нагрузки получается построением модели, аппроксимирующей совместное распределение ее параметров, и является значительно более компактным. Мы проиллюстрируем этот метод на примере [следуя Сринивасану и Клейнману (1974), которые его предложили и использовали при построении модели рабочей нагрузки] установки с пакетной обработкой.

Пример 5.7. Пусть задания характеризуются парой ($t_{\text{ЦП}}$, $n_{\text{в/в}}$) скалярных параметров, где $t_{\text{ЦП}}$ — общее процессорное время, потребляемое заданием, а $n_{\text{в/в}}$ — общее число его запросов на ввод-вывод. Пространством параметров является плоскость, изображенная на рис. 5.9 (a), где каждому заданию соответствует точка. Можно аппроксимировать совместное распределение $t_{\text{ЦП}}$ и $n_{\text{в/в}}$ следующим образом: раз-

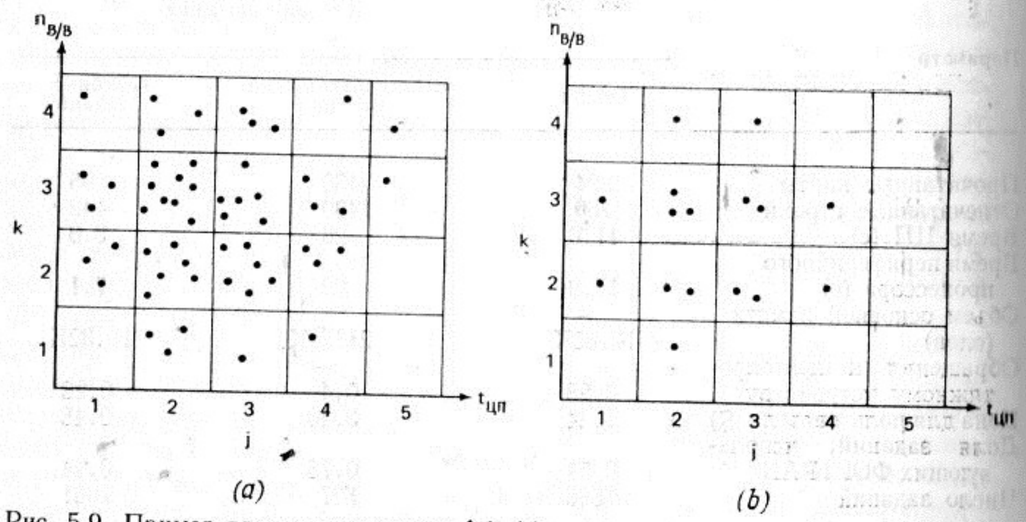


Рис. 5.9. Пример применения метода Ad: (a) — плоскость реальной рабочей нагрузки; (b) — плоскость модели рабочей нагрузки.

делить площадь на одинаковые прямоугольники и сократить число точек в каждом прямоугольнике, разделив это число на общий коэффициент уменьшения. Этот коэффициент близок к отношению между числом заданий в реальной рабочей нагрузке и числом заданий в модели рабочей нагрузки. Результат этой операции показан на рис. 5.9 (b) для коэффициента уменьшения 3.

Как и в методе Ac, значения параметров заданий в модели могут быть выбраны различными способами. Если прямоугольник с центром координат ($t_{\text{ЦП}j}$, $t_{\text{в/в}k}$) на плоскости модели [рис. 5.9 (b)] содержит y_{jk} точек, то мы можем охарактеризовать соответствующие модели заданий y_{jk} парами с теми же значениями ($t_{\text{ЦП}j}$, $t_{\text{в/в}k}$) или y_{jk} парами, соответствующими y_{jk} реальным заданиям, выбранным случайным образом из этого прямоугольника на реальной плоскости [рис. 5.9 (a)]. Могут применяться и другие способы.

Общее число заданий Y в модели может быть определено из проектных спецификаций, таких, например, как максимальное процессорное время, которое разрешено использовать при прогоне модели на данной системе. Мы, например, можем написать:

$$t_{\text{ЦП общ}} = \sum_j \left(t_{\text{ЦП}j} \sum_k y_{jk} \right), \tag{5.12}$$

где $t_{\text{ЦП общ}}$ — общее время центрального процессора, требуемое модели, не может быть больше заданного максимального значения. Если через X обозначить общее число заданий в реальной рабочей нагрузке и через x_{jk} — число реальных заданий в прямоугольнике, определяемом центром ($t_{\text{ЦП}j}$, $n_{\text{в/в}k}$), то коэффициент уменьшения будет

$$r = \frac{X}{Y}, \tag{5.13}$$

где

$$X = \sum_j \sum_k x_{jk}, \quad Y = \sum_j \sum_k y_{jk}, \tag{5.14}$$

и

$$y_{jk} = \frac{x_{jk}}{r} \tag{5.15}$$

для всех j и k.

Нас интересует минимальное значение v, удовлетворяющее проектным требованиям, так как такой выбор соответствует наибольшей степени представительности. Для определения этого минимального значения мы можем использовать простую итеративную процедуру: для начального значения r вычислить $\sum_k y_{jk}$ из (5.15) и $t_{\text{ЦП общ}}$ из (5.12), изменить значение r и повторять эту процедуру до тех пор, пока $t_{\text{ЦП общ}}$ станет достаточно близко к максимальной границе. ■

Хотя и не было проведено серьезного сравнения методов Ac и Ad, последний, вероятно, будет давать бо́льшую степень представительности, особенно когда реальная рабочая нагрузка не может быть разделена на группы с четко определенными границами. Оба метода, как они изложены выше, страдают одним и тем же недостатком. Построение уменьшенного образа реальной рабочей нагрузки основывается на плотности заданий в каждом классе или прямоугольнике, что равносильно приданию всем заданиям одинакового веса. Однако в действительности задания имеют обычно разный вес. Рабочая нагрузка часто определяется в основном несколькими очень большими заданиями, представленными в пространстве параметров изолированными точками, а эти задания могут исчезнуть при применении

коэффициента уменьшения скорее, чем значительно более многочисленные малые задания.

Для исправления этого недостатка оба метода могут быть различным образом модифицированы. Например, можно присвоить соответствующие веса каждому классу или каждому прямоугольнику, так чтобы каждый из них в сущности имел собственный коэффициент уменьшения. В нашей иллюстрации метода Ad (пример 5.7) мы можем присвоить каждому прямоугольнику с координатами ($t_{\text{ЦП}j}$, $n_{\text{в/в}k}$) вес, пропорциональный $t_{\text{ЦП}j}+n_{\text{в/в}k}\cdot\bar{t}_{\text{в/в}}$, где $\bar{t}_{\text{в/в}}$ — среднее время на один запрос ввода-вывода. Изменение метода, учитывающее эту модификацию, оставлено в качестве упражнения для читателя (см. задачу 5.6). Другой подход состоит в том, что прямоугольники не должны быть одинакового размера, при условии что соответствующие прямоугольники на обеих плоскостях совпадают.

Введенные выше веса и другие веса, которые могут быть определены аналогичным образом, на самом деле представляют собой меры, с помощью которых делается попытка как-то количественно выразить относительную важность различных заданий через *объем обслуживания* или *количество ресурсов*, затребованных заданием. Мы видели в разд. 5.2.2, что эту важность следовало бы выразить через ее влияние на производительность и что сложность задачи обычно заставляет нас приближенно описывать ее с помощью индексов, таких, как количество ресурсов. Во всяком случае, демократический принцип приписывания равной важности всем заданиям не годится для вычислительных систем.

Метод Ae может быть использован, когда параметры задания имеют векторный тип и нет оснований использовать метод Aa и Ab. Модель задания, основанная на методе Ae, значительно сложнее модели, основанной на методе Aa. На самом деле это стохастическая модель, в которой каждое задание находится в любой момент времени в одном из возможных состояний, имеющих определенные вероятности переходов в другие состояния (измеренные или оцененные по реальной рабочей нагрузке). Состояние задания определяется текущими значениями его параметров или их ближайшей историей. Вероятности переходов между состояниями зависят от текущего состояния и иногда от тех состояний, в которых задание находилось в недавнем прошлом. Если предположить, что они зависят только от текущего состояния, то мы можем представить задания одной или несколькими марковскими моделями, из которых могут быть определены последовательности значений характеризующих параметров.

Пусть задание характеризуется парой ($\{t_{\text{ЦП}}\}$, $\{t_{\text{в/в}}\}$), где $\{t_{\text{ЦП}}\}$ — последовательность интервалов работы центрального процессора без прерываний (называемых также *процессорными импульсами*) и $\{t_{\text{в/в}}\}$ — последовательность интервалов, блокированных из-за запросов ввода-вывода. Эти последовательности как раз те, что показало бы задание, если бы оно прогонялось одно на системе. Мы можем постулировать,

что интервалы центрального процессора и ввода-вывода генерируются двумя различными и независимыми марковскими процессами, состояний которых есть просто упорядоченные продолжительности интервалов. Если мы разделим ось процессорных импульсов на $q_{\text{ЦП}}$ уровней, то модель импульса центрального процессора состоит из $q_{\text{ЦП}}$ состояний. Переход из состояния $S_{\text{ЦП}i}$ в состояние $S_{\text{ЦП}j}$ произойдет, если за импульсом с длительностью уровня $S_{\text{ЦП}i}$ следует импульс с длительностью уровня $S_{\text{ЦП}j}$.

По-другому и, возможно, точнее можно определить состояния как пары уровней. Если $S_{\text{ЦП}i}$ — уровень процессорного импульса, а $S_{\text{в/в}k}$ — уровень непосредственно предшествующего импульса ввода-вывода, то состояние задания будет определяться парой ($S_{\text{ЦП}i}$, $S_{\text{в/в}k}$), а общее число состояний будет $q_{\text{ЦП}}\cdot q_{\text{в/в}}$ вместо $q_{\text{ЦП}}+q_{\text{в/в}}$ ($q_{\text{в/в}}$ — число интервалов, на которые разбита ось импульсов ввода-вывода). Заметим, что модель (или модели) может быть одной и той же для всех заданий или можно иметь разные модели для разных классов заданий или даже для разных заданий.

Опубликовано очень мало данных, помогающих исследователю в его выборе. Упомянем результаты, полученные Лассетером и др. (1973), которые нашли частично наблюдаемые марковские модели процессорных импульсов, довольно хорошо представляющие поведение около 400 заданий из производственной рабочей нагрузки установки CDC 6600. Эти модели только частично наблюдаемы, так как их состояния соответствуют не различным длительностям, а скорее различным распределениям длительностей.

Модель рабочей нагрузки, полученная методом Ae, описана Линдсеем (1975), охарактеризовавшим рабочую нагрузку установки CDC 6400 двумя однородными марковскими цепями. Одна из них представляет собой цепь *интенсивности входного потока* $s_{\text{вх}}/t_{\text{ЦПвх}}$, где $s_{\text{вх}}$ — число секторов диска, затронутых пересылкой из диска в память, а $t_{\text{ЦПвх}}$ — время, израсходованное процессором между этой пересылкой и следующей того же типа. Другая цепь — интенсивности выходного потока ($s_{\text{вых}}/t_{\text{ЦПвых}}$), где $s_{\text{вых}}$ — число секторов диска, затронутых пересылкой из памяти на диск, а $t_{\text{ЦПвых}}$ — время, израсходованное процессором между этой пересылкой и предыдущей того же типа.

На *шаге B* процедуры построения каждая модель должна быть реализована в исполняемой форме. Модель каждого задания должна быть определена на шаге A одним из рассмотренных выше пяти методов или каким-нибудь другим методом. Имеются три основных подхода к реализации модели задания. Они перечислены в табл. 5.13.

Метод Ba является наиболее простым, если на шаге A v-строки параметров заданий, составляющие модель рабочей нагрузки, извлекаются непосредственно из реальных заданий. Это имеет место в случае метода Ab и, может быть, методов Ac и Ad. Однако при методах Ae и Ad представители каждого класса или прямоугольника могут и не быть реальными заданиями.

Метод Ba может также применяться после использования Aa и Ae, так как мы всегда исследуем реальную рабочую нагрузку для нахождения реального задания, параметры которого ближе всего (согласно математическому или интуитивному определению расстояния) к v-строке, сгенерированной из множества распределений

Таблица 5.13

Методы реализации исполняемой модели каждого задания (шаг B процедуры построения исполняемой модели)

Метод	Описание
Ba	Взять задания из реальной рабочей нагрузки *(эталоны)*
Bb	Написать непараметрические программы или выбрать наиболее подходящие из имеющихся
Bc	Написать параметрические программы *(синтетические задания)*

или матриц перехода. В этих случаях, однако, естественнее и легче использовать метод Bb или Bc.

Реальные задания, выбранные элементами моделей рабочей нагрузки, называются *эталонами*. Эталоны, конечно, являются лучшими моделями для заданий, которые отбираются на шаге A, поскольку они совпадают с заданиями, которые моделируются. Однако их весьма ограниченная изменяемость делает наборы эталонов совершенно негибкими. Самыми легкими изменениями являются, возможно, изменения, рассмотренные Джослином (1965), запретившим некоторые шаги задания (например, компиляцию или выполнение) или генерацию вывода в некоторых эталонах для повышения представительности модели рабочей нагрузки. Другой проблемой, которая может возникнуть с эталонами на некоторых установках, является недоступность реальных заданий из соображений секретности. Это препятствие еще труднее преодолеть из-за того, что файлы, используемые эталонами, должны быть также частью модели рабочей нагрузки.

Метод Bb реализует модели заданий с помощью специально написанных программ, являющихся *непараметрическими*, т. е. такими, чьи запросы на ресурсы не могут быть легко изменены. Эти программы подобны эталонам во всем, кроме того, что они не получаются из реальной рабочей нагрузки. Метод Bb можно использовать вместе с любым методом, рассмотренным для шага A. Им можно заменить метод Ba, когда не существует реальной рабочей нагрузки (например, при разработке новой установки) или когда она недоступна по соображениям секретности.

Трудность написания программ, которые демонстрируют заданное поведение, сильно зависит от степени детализации, с которой это поведение определено. Относительно легко реализовать программу, потребляющую заданное количество времени центрального процессора, но значительно труднее осуществить подгонку к определенному образцу процессорного интервала или к заданному режиму обращений к памяти. В категорию заданий, которые может использовать метод Bb, входят ядра, стандартизированные эталоны и сценарии, равно как и специально написанные программы, соответствующие определенным смесям команд или предложений (см. разд. 5.3.4). Однако в таких случаях при построении модели рабочей нагрузки обычно не придерживаются процедуры, изложенной в этом разделе. Скорее модель собирается по предположениям и суждениям о поведении реальной рабочей нагрузки, которое в подробностях неизвестно. Если реальная рабочая нагрузка детально известна, а представительность является основным проектным требованием, то очень трудно использовать метод Bb с заранее заданным набором программ типа ядер или сценариев.

Метод Bc использует *синтетическое задание* для реализации каждой модели задания. Синтетическое задание — это *параметрическая программа*. Поведением такой программы легко управлять, изменяя значения нескольких входных переменных, называемых аргументами синтетического задания. Это задание имеет простое описание, а реализация синтетического задания, способного воспроизвести поведение разнообразных реальных заданий, тоже довольно проста, как описано в следующем примере.

Пример 5.8. Должна быть построена исполняемая модель рабочей нагрузки для системы с пакетной обработкой. Задания характеризуются парой ($t_{ЦП}$, $n_{в/в}$) однозначных параметров, как в примере 5.7. Шаг A в нашем построении выдает модель рабочей нагрузки в виде определенного числа пар значений $t_{ЦП}$ и $n_{в/в}$ [например, мы можем предположить, что метод Ad был применен и дал результаты для 16 точек на рис. 5.9 (b)]. Задача состоит в построении синтетического задания, по которому будет реализована модель рабочей нагрузки. Эта задача имеет множество решений. При написании нашего синтетического задания мы постараемся сделать его не слишком нереальным.

Предположим для простоты, что в операции ввода-вывода реальных заданий входит только чтение с перфокарт и построчная печать. Так как при пакетной обработке большинство вводов с устройства чтения с перфокарт требуется в начале, а большинство выводов производится в конце, то мы будем строить наше синтетическое задание согласно блок-схеме на рис. 5.10. Аргументы задания обозначены $t'_{ЦП}$, $n'_{в/в}$ и $n'_{вх}$. Мы хотим, чтобы значениями $t'_{ЦП}$ и $n'_{в/в}$ определялись действительные $t_{ЦП}$ и $n_{в/в}$ задания, а $n'_{вх}$ было числом операций ввода. Последний аргумент не нужен, так как наша модель не делает различия между запросами на ввод и на вывод. Однако мы добавили его для повышения гибкости синтетического задания (см. разд. 5.4).

Две переменные k_1 и k_2, которые появляются на блок-схеме (рис. 5.10), должны быть оценены с помощью специальной процедуры или, лучше, измерены; k_1—это время центрального процессора, в течение которого программа находится вне цикла вычислений, а k_2— время выполнения цикла. Заметим, что значение k_1 (которое часто будет очень мало по сравнению с $t'_{ЦП}$) предполагается на рис. 5.10 постоянным, в то время как на самом деле оно зависит от $n'_{в/в}$ и от $n'_{вх}$.

Наше синтетическое задание очень простое и может быть улучшено во многих отношениях. Например, $t'_{ЦП}$, $n'_{в/в}$ и $n'_{вх}$ могут получать начальные значения с помощью программы, а не считываться с перфокарт. Если такое изменение будет сделано, то содержание колоды данных становится совершенно неважным. В нашем

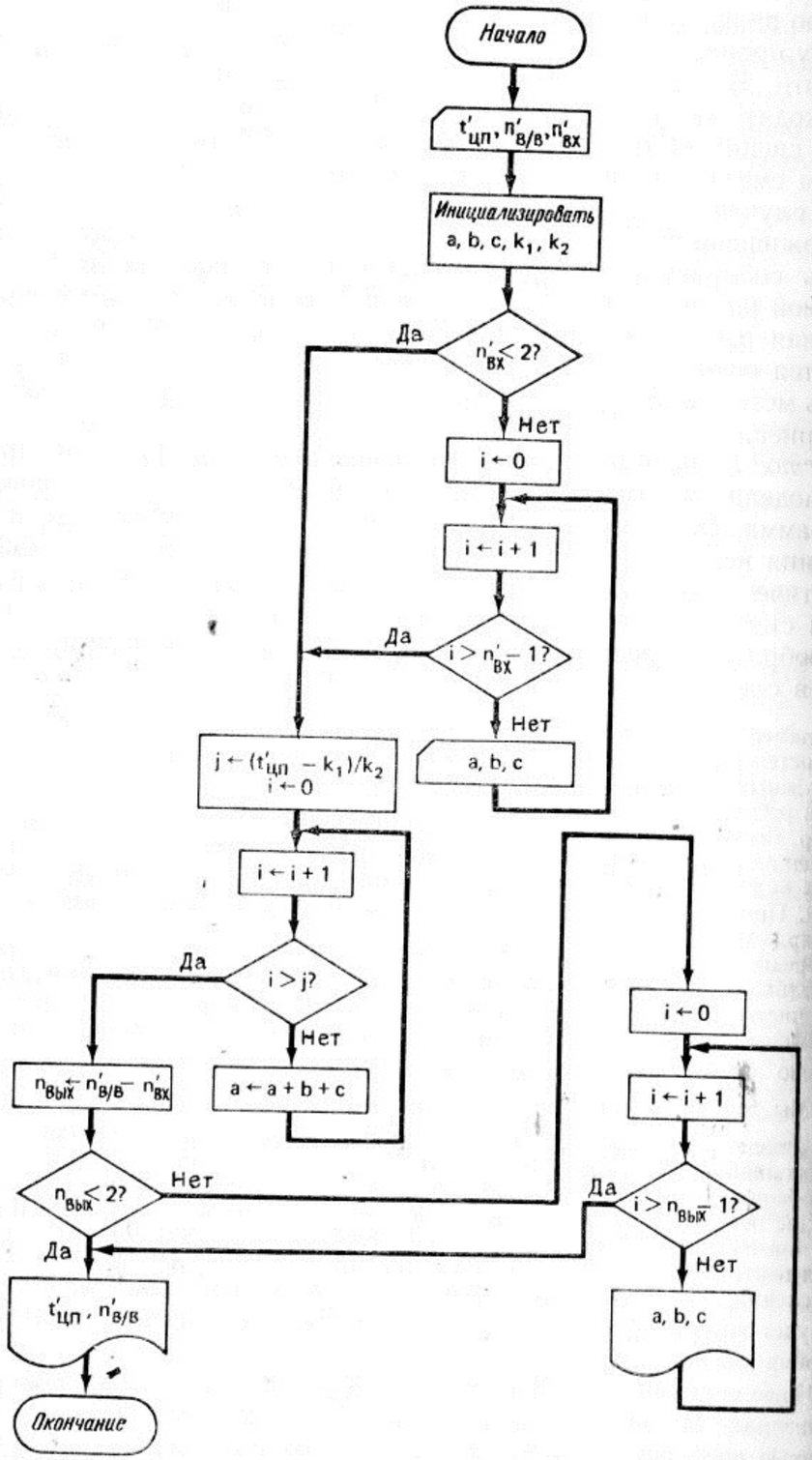


Рис. 5.10. Блок-схема синтетического задания, построенного в примере 5.8.

примере модели рабочей нагрузки на рис. 5.9 (b) мы получим 16 экземпляров синтетического задания, которые соответствуют 16 различным парам ($t_{ЦП}$, $n_{в/в}$), прочитанным с диаграммы на рисунке.

■

Другой пример синтетического задания был предложен Бухгольцем (1969), который ввел этот тип модели задания и предложил использовать его в качестве образца для измерений. На рис. 5.11 приведена блок-схема задания Бухгольца, аналогичного обновляющему файл ядру, описанному в разд. 5.3.4 и на рис. 5.8. [Заметим, что в некотором смысле стандартизированные эталоны, предложенные Госденом и Сиссоном (1962) и Хиллигассом (1966), также представляют собой синтетические задания.] Синтетическое задание Бухгольца имеет три аргумента: число главных записей NMAS, которые должны быть прочитаны, число текущих записей NDET, которые должны быть обработаны, и число выполнений вычислительного цикла NREP при каждом вхождении в этот цикл. Отношения между этими аргументами и количествами ресурсов, потребляемых заданием, менее непосредственны и очевидны, чем те же отношения для простого задания из примера 5.8, но все же достаточно ясны. Трудность вычисления значения аргументов задания, которые будут соответствовать строке параметров задания, полученных на шаге A, зависят от выбранного описания рабочей нагрузки. Если модель задания так же проста, как и принятая в примере 5.8, то это вычисление будет относительно простым (см. задачу 5.13).

Синтетические задания, аналогичные предложенному Бухгольцем, использовались несколькими исследователями. Например, Вуд и Форман (1971) только слегка изменили его при построении своей модели рабочей нагрузки методами Ab и Bc (метод Ba не мог быть использован из соображений секретности). Сринивасан и Клейнман (1974) добавили три аргумента: число повторений обновления обрабатываемого файла, размер буфера ввода-вывода и длину записи. Значения параметров задания $t_{ЦП}$ и $n_{в/в}$, полученные применением метода Ad, должны быть переведены в значения шести аргументов. В общем случае это нетривиальная задача. Однако, как будет показано в разд. 5.4, степени свободы, полученные за счет использования большего числа аргументов, чем число параметров задания, можно использовать для калибровки модели рабочей нагрузки. Немного более сложное, чем в примере 5.8, синтетическое задание было использовано Керниганом и Гамильтоном (1973) в качестве основного компонента их модели рабочей нагрузки для машины Honeywell 6070 под управлением операционной системы GECOS III. В качестве аргументов синтетического задания были выбраны общее процессорное время, общее время ввода-вывода и занимаемый объем основной памяти. Объем памяти, запрашиваемый каждым заданием, предполагается независимым от времени.

Следует заметить, что метод Bc — это единственный метод, который в сочетании с методом Aa или Ae позволяет генерировать ис-

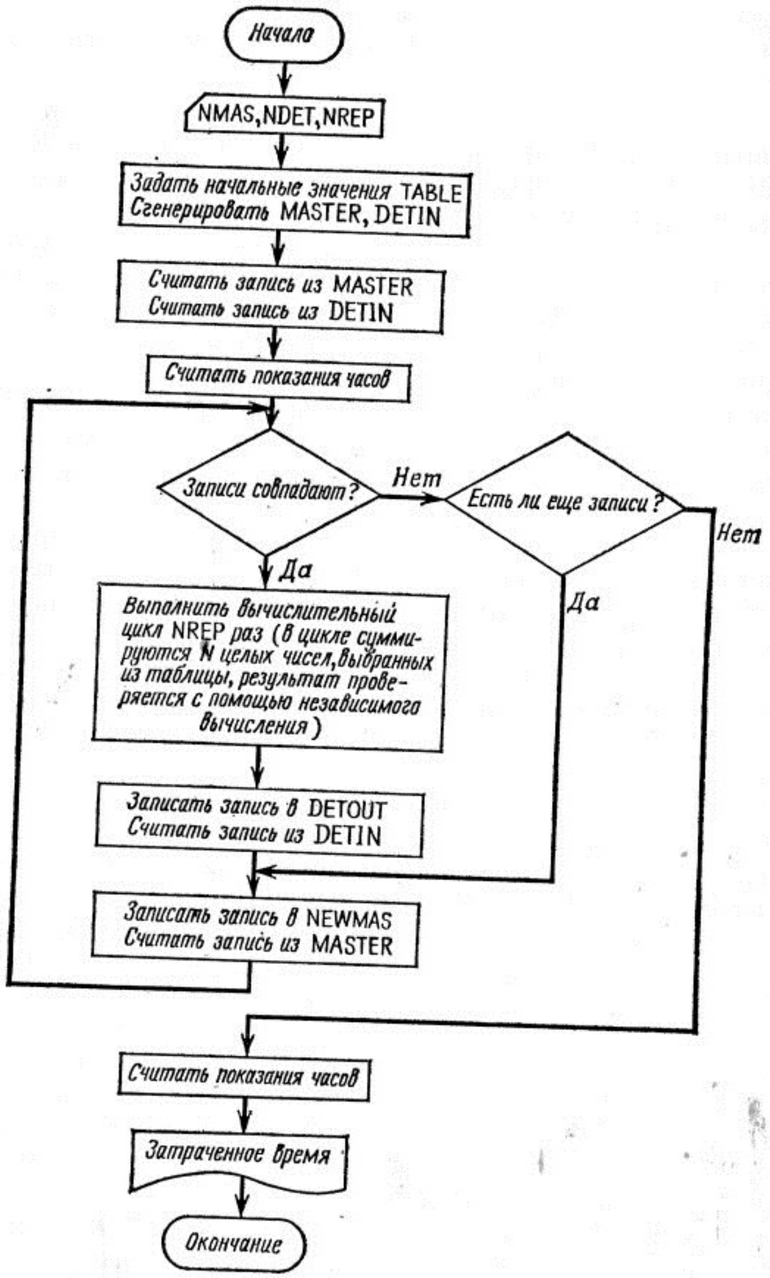


Рис. 5.11. Блок-схема синтетического задания, предложенного Бухгольцем (1969). Записи в MASTER имеют последовательные ключи; ключи записей в DETIN следуют с приращением, равным отношению NMAS/NDET. TABLE — массив размерности N^3. Таким образом, N влияет и на время выполнения вычислительного цикла, и на объем занимаемой программой памяти.

пользуемую модель рабочей нагрузки в реальном времени, т. е. в том времени, в каком нужно. Это можно сделать с помощью выборки из заданных распределений или используя стохатические модели для генерации значений параметров задания в момент его поступления. Момент поступления также может быть определен выборкой из заданного распределения времен между поступлениями. Сгенерированные таким образом параметры задания передаются в копию синтетического задания, становящуюся моделью задания, которая должна быть немедленно введена в систему. При этом подходе важно, чтобы соотношения между параметрами задания и аргументами синтетического задания были как можно проще.

Читатель, конечно, заметил близкое подобие этого метода генерации модели рабочей нагрузки в реальном времени и метода, использовавшегося при моделировании, управляемом распределением (см. разд. 3.2.4). Единственные концептуальные отличия состоят в том, что в измерительных исследованиях этот метод применяется к реальной системе, а не к ее модели, а также в том, что поскольку модель рабочей нагрузки должна быть исполняемой, то необходимо посредничество синтетических заданий.

Этот метод генерации модели рабочей нагрузки, дающий очень компактную модель, был применен Шветманом и Брауном (1972) и Линдсеем (1975).

Сильнейшим аргументом в пользу синтетических заданий является их гибкость, которая может быть использована при калибровке модели рабочей нагрузки, а также и при ее модификациях. Кроме того, если только различные классы заданий реальной рабочей нагрузки не были описаны по-разному, то обычно в модели достаточно иметь только несколько разных синтетических заданий. Таким образом, получающаяся в результате модель часто более компактна, чем модель, полученная методами Ba и Bb, особенно тогда (как отмечено выше), когда исполняемая модель генерируется в реальном времени. Оливер и др. (1974), например, доказывают, что их библиотека из пяти синтетических заданий, ориентированных на пять различных типов коммерческих применений системы, может представлять при подходящей комбинации заданий очень широкий спектр коммерческих рабочих нагрузок.

Синтетические задания одинаково удобно применять вместе со всеми методами, описанными на шаге A. Главным недостатком синтетических заданий по сравнению с эталонами является присущая им меньшая степень представительности. При одинаковых описаниях задания удачно выбранные эталоны, вероятно, представляют реальные применения установки лучше, чем синтетические задания. Последние прежде всего требуют более точного описания, чтобы достичь той же потенциальной представительности, что и у эталонов. Однако в некоторых исследованиях сообщаются обнадеживающие результаты об использовании синтетических заданий даже с очень простым описанием.

Мы завершим этот раздел кратким обсуждением способов, которыми исполняемую модель рабочей нагрузки можно представить измеряемой системе. В установках с пакетной обработкой модель рабочей нагрузки часто физически состоит из некоторого числа колод перфокарт или из файла на ленте или на диске. Если для реализации модели был применен метод Вс, то синтетическое задание (или задания) можно также вызвать *диспетчером*. Диспетчер представляет собой программу, считывающую параметры задания с устройства чтения перфокарт или некоторого другого входного устройства или генерирующую эти параметры из заданного распределения или стохастической модели. Потом диспетчер вычисляет по этим параметрам значения аргументов синтетического задания, передает эти значения в копию задания и моделирует поступление задания (т. е. ставит его в очередь). Как было отмечено выше, времена поступления могут планироваться методом очередного события, используемым при моделировании, если распределение времен между поступлениями является частью модели рабочей нагрузки (см. пример 5.6).

Когда диспетчер представляет собой программу, прогоняемую на измеряемой системе, мы назовем его *внутренним диспетчером*. Если модели заданий генерируются и представляются системе другой вычислительной системой, подключенной к ней в качестве периферийного устройства, то мы имеем *внешний диспетчер*. Таким образом, внешний диспетчер представляет собой комплекс программ и оборудования. До сих пор диспетчеры довольно редко использовались в установках с пакетной обработкой, и практически все они внутреннего типа.

В диалоговых системах единственная альтернатива диспетчеру для эксперимента с использованием искусственной модели рабочей нагрузки состоит в привлечении стольких людей, сколько терминалов должны быть активными. Эти люди должны играть роль пользователей, следуя сценариям, выданным экспериментатором. Поскольку такая альтернатива очень неудобна и значительно понижает воспроизводимость модели рабочей нагрузки, обычно предпочитают решение с диспетчером. Внутренние диспетчеры не требуют дополнительного оборудования, но вмешиваются в измеряемую систему. Более того, они не могут моделировать без искажений появления команд с терминала и выводы из-за того, что они минуют часть путей ввода-вывода системы. Этого нет в случае внешнего диспетчера, который может более точно представить переговоры между системой и терминалами. Одной из проблем (как видно из примера 5.6) для внутреннего диспетчера является проблема перехвата всех сообщений, посылаемых на терминалы, и моделирование их пересылки и приема пользователями, а также моделирование времени обдумывания и времен ввода-вывода сообщений. Заметим, что внутренние диспетчеры иногда вместо последовательностей диалоговых команд моделируют запросы на ресурсы. Диспетчеры обоих типов и генерируемые ими модели

могут прогоняться одновременно с реальными процессами и не обязательно требуют того, чтобы им была отдана вся система.

Описания внутренних диспетчеров можно найти у Де Мейса и Вейзера (1969), Зальцера и Джинтелла (1970) и Фогеля и Винограда (1972). Примером внешнего диспетчера является диспетчер, разработанный Гринбаумом (1969) для системы Multics: мини-машина PDP-8, связанная через телефонные каналы с вычислительной системой GE-645 может моделировать одновременно до 12 диалогов с

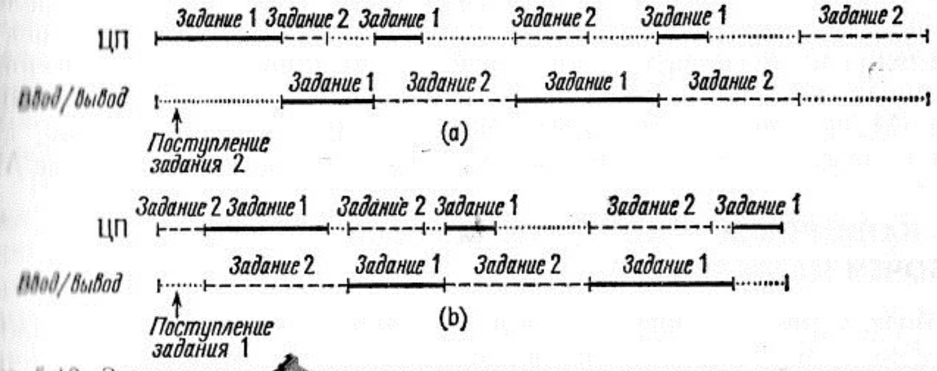


Рис. 5.12. Зависимость общего затраченного времени от порядка появления заданий в мультипрограммной системе: (a) — задание 1 предшествует заданию 2; (b) — задание 2 предшествует заданию 1. Точками показаны интервалы, в которых данный ресурс не используется.

пользователями, команды которых поступают по сценарию, как это показано в табл. 5.9.

Одной из проблем, требующих решения всякий раз, когда модель рабочей нагрузки состоит из последовательности моделей заданий, разработанных экспериментатором, является проблема определения порядка, в котором задания должны появляться в последовательности. Проще говоря, порядок обработки заданий влияет на перекрытия вычислений и обслуживание ввода-вывода и, следовательно, на такие индексы производительности, как пропускная способность или время ответа. Поэтому различные стратегии планирования дают разные производительности, и поведение данного метода в общем случае зависит от порядка и от моментов поступления заданий.

Простой иллюстрацией этого факта (см. рис. 5.12) является случай двух заданий, конкурирующих за центральный процессор и канал ввода-вывода. Когда задание *1* предшествует заданию *2*, то общее потраченное время существенно больше, чем в противном случае. Отметим, что различие между потраченными временами в значительной части объясняется *хвостовым эффектом*, рассмотренным в разд. 3.4.2.

Даже при определении порядка поступления заданий экспериментатор должен пытаться максимизировать представительность рабочей нагрузки. Если в установке приняты специальные стратегии, влияющие на этот порядок, или если реальная рабочая нагрузка

проявляет определенные особенности появления (например, большие задания имеют тенденцию поступать в конце дня), то модель должна их отражать. В большинстве установок задания имеют внешние приоритеты, влияющие на порядок, в котором они обрабатываются. Моделям заданий в этих установках должны быть назначены внешние приоритеты согласно наблюдаемым частотам и образцам таких приоритетов. Стохастические модели процессов поступления могут вырабатываться на основе эмпирических данных и использоваться для обеспечения достаточно представительной упорядоченности моделей заданий. Когда недоступна никакая информация об этом аспекте поведения коллектива пользователей, то для уменьшения искажений, вносимых исследователем или методом генерации модели задания (шаг А), нужно применять процедуру рандомизации. Читатель, конечно, заметил, что такая процедура была уже построена в методе Ab.

5.4. КАЛИБРОВКА И КОНТРОЛЬ МОДЕЛИ РАБОЧЕЙ НАГРУЗКИ

Модель рабочей нагрузки, как и всякая другая модель, должна быть по возможности откалибрована и проконтролирована. Различие между этими двумя операциями уже обсуждалось в разд. 3.3 и 4.4. Обе они основаны на сравнении, которое, как предполагается в случае модели рабочей нагрузки, указывает степень соответствия модели реальной рабочей нагрузке. Согласно определению представительности, данному в разд. 5.2.2, мы будем сравнивать значения индексов производительности, выдаваемых двумя рабочими нагрузками, чтобы определить, насколько они близки к совпадению. Конечно, это самое надежное сравнение, если модель рабочей нагрузки исполнима, а реальная рабочая нагрузка и система доступны. Когда эти условия выполнены, то возможны экспериментальные сравнения, но эксперименты должны быть проведены с чрезвычайной заботой об обеспечении одинаковых условий в системе, управляемой этими двумя рабочими нагрузками. Если выполнены не все условия, то возможно только сравнение концепций (см. разд. 3.3.1).

Когда модель рабочей нагрузки неисполняемая, она будет управлять моделью системы и на результаты экспериментов по калибровке и контролю будет очень сильно влиять точность модели системы. В действительности, так как модель системы при калибровке и контроле требует пригодной модели рабочей нагрузки, единственным разумным решением представляется проведение обеих этих операций одновременно. Часто это решение вынужденное из-за того, что две модели не могут быть легко и аккуратно разделены. Конечно, могут быть ситуации, когда модель системы известна или о ней можно с уверенностью предположить, что она пригодная. К тому же, когда имеется некоторая свобода изменений системы без изменения модели рабочей нагрузки, то мы ее можем использовать для исследования степени независимости модели рабочей нагрузки от системы. Однако

наши рассмотрения в этом разделе относятся в основном к исполняемым моделям рабочей нагрузки.

Для того чтобы представить приемлемо большую область пригодности, модели рабочей нагрузки, как и модели систем, должны быть более детализированными, чем это нужно для их представительности при нескольких экспериментальных условиях. Эту дополнительную степень детализации можно ввести не прямо, а расширив набор индексов производительности, сравниваемых при калибровке и контроле. Другими словами, модель рабочей нагрузки, которая может давать достаточно точные значения не только основных индексов производительности, но и вторичных, имеет, возможно, более широкую область пригодности, чем модель, на которой можно проверить только основные индексы.

Шветман и Браун (1972), например, провели контроль своей модели рабочей нагрузки для CDC 6600, сравнивая не только средние пропускные способности и коэффициенты использования центрального процессора, которые были их основными индексами, но и коэффициенты использования разных дисковых каналов, средние интенсивности запросов периферийных процессоров и операционной системы и коммутации заданий. Это сравнение было сделано при прогоне модели рабочей нагрузки, составленной из синтетических заданий, для конфигурации системы, воспроизводящей как можно более верно для этих заданий условия обработки производственной рабочей нагрузки. Так как основное синтетическое задание не включает некоторые действия по пакетной или диалоговой работе ввода-вывода, то ресурсы, обычно потребляемые этой работой (часть основной памяти, один из путей, доступных для загрузки заданий в память, и четыре периферийных процессора), были отключены. Другой пример — работа Кернигана и Гамильтона (1973), опиравшихся в своем сравнении на 13 индексов. Эти индексы включали пользовательское время центрального процессора, системное процессорное время, времена ввода-вывода для каждого устройства вспомогательной памяти, номера операций ввода-вывода, не занятые интервалы центрального процессора, попеременную загрузку программ.

Сравнение, показывающее неполное совпадение, должно привести к изменению модели. Изменения, особенно при калибровке модели, должны быть по возможности несложными. Модель рабочей нагрузки в идеале должна строиться, исходя из ясного знания наиболее вероятных изменений, которые предстоит внести. Так как и предсказание этих изменений, и хорошее использование этой информации — довольно сложная вещь, обычно нужно строить достаточно гибкие модели, что уменьшает вероятность их переделки. Среди исполняемых моделей самыми гибкими являются модели, составленные из синтетических заданий. Калибровка модели рабочей нагрузки такого типа может заключаться в изменении аргументов некоторых ее синтетических заданий, для того чтобы сделать модель более представительной. К тому же возможно ввести при построении синтетического

задания некоторые особые *калибровочные параметры*, которые будут, очевидно, первыми кандидатами на изменение. Примерами таких параметров синтетического задания на рис. 5.10 могут быть k_1 и k_2 или, что наверное правильнее (так как k_1 и k_2 использовались как корректировочные множители), $n'_{вх}$ — третий аргумент.

В принципе можно распространить методы, используемые для калибровки моделей, на калибровку синтетических моделей рабочей нагрузки. В частности, описанный в разд. 3.3.3 метод определения изменений, которые нужно сделать, может быть применен и к этой задаче. Как построение модели рабочей нагрузки, так и калибровка соотношений между аргументами синтетического задания и характеризующими их параметрами должны быть возможно проще. Идеальным будет тот случай, когда каждый параметр зависит простым образом только от одного аргумента. Тогда можно обособить каждый компонент поведения системы, а вычисление значений аргументов и их изменения в процессе калибровки становятся много проще.

Пример 5.9. Исследуем соотношения между аргументами синтетического задания, построенного в примере 5.8 (см. рис. 5.10), и его поведением. В идеале два компонента поведения задания, $t'_{ЦП}$ и $n_{в/в}$, довольно хорошо разделены и прямо связаны с аргументами $t'_{ЦП}$ и $n'_{в/в}$. Изменения $t'_{ЦП}$ влияют только на $t_{ЦП}$, а не на $n_{в/в}$. Значение $n_{в/в}$ в точности равно $n'_{в/в}$, которое, однако, некоторым образом влияет на $t_{ЦП}$. Для определения меры этого влияния, равно как и для контроля нашего проекта, мы можем проделать несколько экспериментов с синтетическими заданиями и изобразить измеренные значения $t_{ЦП}$ и $n_{в/в}$ в зависимости от изменения $t'_{ЦП}$ (при постоянном $n'_{в/в}$) и в зависимости от $n'_{в/в}$ (при постоянном $t'_{ЦП}$). Присутствие $n'_{вх}$ в аргументах делает нашу задачу трехмерной. Однако в данном случае мы будем игнорировать этот аргумент, так как учитывается он очевидным образом, но это сделает наше рассмотрение более громоздким. Цели нашего проектирования могут быть выражены двумя уравнениями:

$$t_{ЦП} = t'_{ЦП}, \quad n_{в/в} = n'_{в/в}. \tag{5.16}$$

Если эксперименты и сбор данных были проведены достаточно внимательно, то кривые, о которых говорилось, покажут нам, насколько отличается наше синтетическое задание от этих целей. В случае существенных расхождений мы можем также использовать эти кривые в целях калибровки. Мы можем, например, брать из этих кривых значение $t'_{ЦП}$ и подставлять его в синтетическое задание, чтобы получить заданное значение $t_{ЦП}$.

Если зависимость $t_{ЦП}$ и $n_{в/в}$ от $t'_{ЦП}$ и $n'_{в/в}$ очень запутана, то подгонка значений $t'_{ЦП}$ и $n'_{в/в}$ для получения желаемых значений $t_{ЦП}$ и $n_{в/в}$ может оказаться очень сложной. Примером может служить работа Сринивасана и Клейнмана (1974), в которой у синтетического задания имеется шесть аргументов и два характеризующих параметра, нелинейно зависящие от аргументов.

Набор синтетических заданий, в которых достигнута высокая степень разделения параметров, описан Оливером и др. (1974). Эти авторы сообщают о результатах некоторого числа экспериментов с отдельными синтетическими заданиями. Эксперименты очень важны в особенности из-за того, что трудно предсказать и соответствующим образом принять во внимание влияния на параметры задания остальной рабочей нагрузки (в идеале этого влияния не должно быть) и операционной системы. Более того, если синтетические задания не написаны на языке машины, различные трансляторы оказывают в общем случае труднопредсказуемое влияние на взаимосвязь между

аргументами и параметрами задания. Во всех этих случаях важно провести эксперименты.

■

Можно было бы предложить метод построения модели рабочей нагрузки, основанный на процедуре калибровки. Этот метод заключался бы в итеративной подгонке аргументов, экспериментах и новой подгонке и в последовательном добавлении копий синтетических заданий с последующим отбором их аргументов и новыми экспериментами. Это предложение, хотя и привлекательное, не представляется в настоящее время широко применимым, так как правила и критерии, которыми следует руководствоваться при принятии решений, неизвестны и потребуется, вероятно, слишком много дорогостоящих экспериментов.

Всякий раз, когда в систему вносятся изменения или берется другая система, должен выполняться контроль модели рабочей нагрузки. Эту операцию, однако, может быть очень сложно или вовсе невозможно произвести. Даже в случае небольшого изменения системы, проверяемого до его реализации, нет способа измерить индексы производительности измененной системы при реальной рабочей нагрузке. Это можно сделать только после реализации этой нагрузки пользователями.

Когда реальная рабочая нагрузка проектируется, ее нельзя использовать для управления системой в целях сравнения. Проблемы, возникающие при сравнении различных систем, еще более сложны. Таким образом, почти во всех практически интересных случаях пригодность модели рабочей нагрузки должна быть постулирована. Важно, чтобы модель по своей природе была пригодна для столь широкой области систем и конфигураций, какая нужна для данного исследования по оценке. Эта проблема рассматривалась в разных местах данной главы, в частности ранее в этом разделе. Однако мы должны признать, что наши знания в этой области, равно как некоторые другие аспекты описания рабочей нагрузки, все еще находятся на довольно низком уровне.

Задачи

5.1. Составьте смесь команд, представляющую рабочую нагрузку, обработка которой на машине А на 10% быстрее, чем на машине В (табл. 5.6).

5.2. Должна быть выбрана система для установки, на которой все программы пользователей написаны на одном языке (например, на ФОРТРАНе). Нужно сравнить производительность нескольких пар (аппаратное обеспечение, компилятор). Опишите шаг за шагом, как вы будете решать эту задачу, используя смесь команд (например, из табл. 5.7) для описания рабочей нагрузки установки. Установите, какие измерительные эксперименты вы проведете и как вы будете их строить, включая описание используемых измерителей. Сравните, особенно с точки зрения воспроизводимости результатов, преимущества и недостатки этого подхода и подхода, основанного на описании модели рабочей нагрузки с помощью физических ресурсов.

5.3. Определите три типа изменений ПРОП-системы пакетной обработки, представленной на рис. 3.6, относительно которых описание рабочей нагрузки системно-независимо, и три типа, относительно которых оно системно-зависимо. Можете ли вы

придумать описание, независимое относительно последних трех типов изменений или всех вместе?

5.4. Нужно провести исследование по определению возможности управления диалоговой системой с помощью набора вероятностных моделей поведения пользователя (метод Ae из разд. 5.3.5). Управление должно быть организовано по схеме, показанной в примере 5.6, но исполняемая трассировочная запись будет заменена вероятностной моделью, которая будет генерировать в реальном времени следующую команду для каждого пользователя. Модифицируйте блок-схему на рис. 5.6 (c) и постройте измерительные эксперименты, которые мы провели бы для сбора информации, требуемой диспетчеру.

5.5. Который из методов из табл. 5.10 дает, по вашему мнению, наивысшую потенциальную представительность? Почему? Какой метод дает, при прочих равных условиях, наиболее системно-независимую модель рабочей нагрузки?

5.6. Измените итеративную процедуру, определяющую коэффициент уменьшения в методе Ad (см пример 5.7), чтобы она учитывала введение веса, пропорционального $t_{\text{ЦП}j}+n_{\text{в/в}k}\bar{t}_{\text{в/в}}$ для каждого прямоугольника с координатами центра ($t_{\text{ЦП}j}$, $n_{\text{в/в}k}$). Заметим, что $\bar{t}_{\text{в/в}}$ — среднее время на запрос ввода-вывода. Можете ли вы получить подобный результат, выбирая прямоугольники переменных размеров вместо того, чтобы давать им разные веса? Как вы будете выбирать размеры прямоугольников?

5.7. Вы воспользовались методом Ac для классификации заданий реальной рабочей нагрузки. Как вы присвоили бы веса разным классам, чтобы получить более представительную модель, чем модель с равными весами заданий? Предположим, что для каждого задания выбрано описание:

(a) пара ($t_{\text{ЦП}}$, $n_{\text{в/в}}$),

(b) тройка ($t_{\text{ЦП}}$, $t_{\text{в/в}}$, m), где $t_{\text{в/в}}$ — общее время ввода-вывода, включая время ожидания в очереди, а m — объем памяти, запрошенный заданием.

5.8. Можно ли применить метод Ac или Ad для описания, содержащего параметры векторного типа? Если да, то как?

5.9. Постройте синтетическое задание с такими же аргументами, как в примере 5.8, сделав $n'_{\text{вх}}$ всегда равным $n'_{\text{в/в}}/2$, при структуре задания, представляющей собой единственный цикл, повторяемый $n'_{\text{вх}}$ раз. Цикл состоит из считывания одной карты, вычисления, продолжающегося определенное время, и печати одной строки. Прокомментируйте взаимосвязь между $t'_{\text{ЦП}}$ и $t_{\text{ЦП}}$ и между $n'_{\text{в/в}}$ и $n_{\text{в/в}}$ после введения в ваше задание соответствующих поправок, если они нужны.

5.10. Должна быть построена исполняемая модель рабочей нагрузки для ПРОП-системы пакетной обработки, отличающейся от системы на рис. 3.6 отсутствием барабана. Каждый запрос на ввод (чтение с перфокарт) приводит к вызову подпрограммы ввода. Эта подпрограмма ищет в буфере ввода запрошенные данные. Если буфер пуст, подпрограмма приостанавливает задание и запрашивает сектор диска, содержащего очередные входные данные для обработки, для пересылки в буфер. Завершение этой пересылки приводит к тому, что задание становится готовым к дальнейшему прогону. Каждый запрос на вывод (печать) ведет к вызову подпрограммы вывода, которая записывает в буфер вывода данные для печати и возвращает управление заданию. Когда буфер полон, подпрограмма приостанавливает задание и пересылает содержимое буфера вывода на диск. Завершение этой операции приводит задание в состояние готовности. Размер каждого буфера (следовательно, и сектора диска) эквивалентен 25 картам или 15 строкам. Интересующий нас индекс производительности — пропускная способность. Проверка представительности, используемая в примере 5.8 и задаче 5.9, состоит в рассмотрении того, нельзя ли пренебречь влиянием на среднюю пропускную способность того порядка, в котором каждое задание производит операции ввода и вывода. Предположим, что общее время ожидания диска, исключая ожидания в очереди, постоянно и равно 10 мс, что два одинаковых синтетических задания с характеристиками $t'_{\text{ЦП}}=100$ мс, $n'_{\text{в/в}}=300$, $n'_{\text{вх}}=150$ поступают одновременно в пустую систему и что никогда не возникает задержек диска из-за работы устройства чтения с перфокарт или печатающего устройства. Сравните среднюю пропускную способность, даваемую двумя заданиями типа рассмотренного в примере 5.8, с пропускной способностью двух заданий типа рассмотренного в задаче 5.9 (с чередующимися вводами и выводами). Целесообразно решить задачу сначала с одним заданием. Если сравнение покажет, что разницей в пропускных способностях можно пренебречь, будет ли это достаточным основанием считать, что данное описание имеет удовлетворительную степень представительности? Обсудите также предположение, что влиянием на пропускную способность пересылок с устройства чтения перфокарт на диск и с диска на печатающее устройство можно пренебречь.

5.11. Оценочное исследование должно определить влияние размера буфера ввода-вывода на производительность ПРОП-установки, описанной в задаче 5.10. Независимость от системы описания задания парой ($t_{\text{ЦП}}$, $n_{\text{в/в}}$) нужно оценить, используя первое или второе задание из задачи 5.10 (или оба задания). Вновь предполагается, что исследование в однопрограммной среде предшествует исследованию в мультипрограммной среде, состоящей из двух одинаковых синтетических заданий. Аргументы заданий могут быть теми же, что и в задаче 5.10. Является ли и до какой степени критичным выбор их значений?

5.12. Предполагая, что распределения, использованные Шветманом и Брауном (1972), являются подходящими (см. описание метода Aa в разд. 5.3.5), определите соответствующееся с ними описание задания и нарисуйте блок-схему задания, основывающегося на этом описании. Разработайте также процедуру (в форме блок-схемы), которая работала бы совместно с диспетчером, аналогичным диспетчеру Шветмана и Брауна, чтобы генерировать значения аргументов для каждой копии синтетического задания по заданным распределениям.

5.13. Покажите, как можно вывести значения аргументов NMAS, NDET, NREP синтетического задания, предложенного Бухгольцем (1969), из значений параметров ($t_{\text{ЦП}}$, $t_{\text{в/в}}$, m, которые могут использоваться для описания заданий из моделируемой реальной рабочей нагрузки. Предполагается, что четыре файла, связанные с синтетическим заданием, записаны на четырех лентах, установленных на четырех лентопротяжных устройствах, каждое со своим независимым контроллером.

5.14. Возможно ли и полезно ли реализовать синтетический сценарий, т. е. сценарий, который сможет заключать в себе широкий спектр поведения терминальных пользователей, просто путем изменения значений нескольких аргументов? Какие аргументы вы бы выбрали, например, если бы реализовали подобным образом сценарий, приведенный в табл. 5.9?

5.15. Некая модель рабочей нагрузки получена применением методов Ad и Bc. Как вы отнесетесь к тому, чтобы калибровать ее с помощью экспериментального сравнения даваемых ею индексов производительности с индексами производительности модели, полученной точно так же, за исключением того, что вместо метода Bc использовался метод Ba?

6
Оценка производительности в задачах выбора

6.1. ЗАДАЧИ ВЫБОРА

В разд. 1.5 работы по оценке производительности были разделены на три главные категории: выбора, усовершенствования и проектирования. Работы по выбору были определены как такие, которые направлены на выработку технической информации, необходимой пользователям для принятия решений, как удовлетворить их запросы на обработку информации посредством выбора одной из нескольких возможностей. Таким образом, выбор есть характерное действие потребителей, тогда как проектирование есть задача поставщика или изготовителя. Мы уже отмечали в разд. 1.5, что это различие между типами задач оценки не является однозначным и проводится только для удобства объяснения. Подобно большинству классификаций в этой книге, этим следует пользоваться с осторожностью. Действительно, некоторые конструкторские разработки обычно выполняются организациями, которые не производят и не продают вычислительных систем, компонентов или программного обеспечения, а в основном являются пользователями машин. Например, на некоторых установках имеются системные компоненты (аппаратура или модули операционных систем), которые были разработаны и изготовлены обслуживающим персоналом. Кроме того, большинство изготовителей вычислительных машин приобретают оборудование (например, периферийные устройства), а иногда и программное обеспечение у других изготовителей. Таким образом, они действуют как покупатели по отношению к так называемому рынку первичных изготовителей оборудования и встречаются с типичными проблемами выбора.

Даже граница между выбором и усовершенствованием определена недостаточно четко. При попытке усовершенствования может обнаружиться, что наилучшим направлением было бы расширить систему, добавив к ней определенное устройство, например дисковод. Очевидно, что в результате возникает проблема выбора дисковода. Заметим, что эта задача весьма напоминает задачу выбора дисковода для проектируемой установки при условии, что ее конфигурация уже определена. Однако в последнем случае мы располагаем большей свободой, так как легче модифицировать проектируемую конфигурацию, чем уже смонтированную.

Ниже для целей нашего изложения мы произведем классификацию наиболее часто встречающихся проблем выбора по следующим категориям.

a) *Выбор режима обработки.* Среди вопросов, на которые следует иметь ответ, когда установка будет смонтирована, некоторые касаются режима (режимов) обработки, который нужен (или наиболее предпочтителен) для коллектива пользователей. Основные типы: пакетный, диалоговый, наличие или отсутствие доступа пользователей к машине, прямой доступ и работа в реальном времени. Выбор режима обработки позволяет разработчикам существенно ограничивать области приемлемых решений аппаратуры и программного обеспечения.

b) *Выбор поставщика.* Это центральная проблема в создании установки. Существуют серьезные различия между совершенно новой установкой (например, машинной информационной системой, вводимой в некоторой организации) и приобретением новой системы для существующей установки. Однако во всех случаях экономические и технические ограничения часто сужают выбор до ограниченного числа альтернатив, способность которых удовлетворить требованиям производительности еще предстоит оценить. В конкурсном выборе эти альтернативы предлагаются различными поставщиками, которые разрабатывают одну или несколько конфигураций, используя свои компоненты оборудования и программного обеспечения. В этом состоит причина для такого названия этой категории проблем, которая в действительности заключается в выборе одной из нескольких систем-кандидатов. Разработка конфигурации, которая обычно включает выбор компонентов, будет обсуждаться в гл. 8.

c) *Выбор установки.* Это категория проблем, которые должны быть решены, если предстоит выбрать наиболее подходящую установку из нескольких имеющихся, чтобы удовлетворить потребность в обработке информации. Организация может располагать различным вычислительным оборудованием, и отдел или группа, или отдельный программист в этой организации может оказаться перед необходимостью решать, какое именно оборудование наиболее желательно для данного применения. Другим примером является выбор обслуживающей организации (компании или организации, продающей машинное время) для обработки заданной рабочей нагрузки. В узко техническом смысле эти проблемы похожи на выбор поставщика. Однако их экономические аспекты и различие подходов (в этом классе — только подход пользователя) может сделать общую картину совсем иной.

d) *Выбор компонентов системы.* К этому классу мы относим проблемы развития или расширения системы, мотивированные требованиями или пожеланиями улучшения производительности. Они характеризуются тем, что компонент или компоненты, которые предстоит выбрать, должны быть добавлены к существующей системе в рамках существующей установки. Остальные проблемы этого класса относятся к выбору при проектировании конфигурации. В соответствии с нашим определением вычислительной системы мы отнесем к этой категории выбор операционных систем или их модулей (включая пакеты программ, программные измерительные средства и, если они счи-

таются частью системы, компиляторы, программы обслуживания баз данных, вспомогательные программы) наряду с модулями оборудования (блоками основной памяти, центральным процессором, процессорами ввода-вывода, дисководами, лентопротяжными устройствами, мультиплексорами и т. п.).

e) *Выбор прикладной программы.* Выбор такого типа обычно осуществляется пользователем, а не администрацией установки (в отличие от случая модулей программного обеспечения в п. d). Разумеется, некоторые из критериев, на которых основан такой выбор, возможно, совпадают в этих двух случаях. Различие между этими случаями на самом деле нечетко, так как оно зависит от принятого определения границ системы. Таким образом, выбор компилятора или системы обслуживания базы данных попадет в одной ситуации в класс d, а в другой ситуации в этот класс. Приведем примеры проблем, относящихся к данному классу:

a) выбор одного из нескольких представленных на установке языков для кодирования данной программы;

b) выбор одного из существующих на установке пакетов программ для решения данной задачи (например, генерация псевдослучайных чисел, дисперсионный анализ, решение уравнений в частных производных, сортировка файла);

c) выбор одного из имеющихся на рынке пакетов программ для удовлетворения определенных запросов на обработку информации (например, анализ программ на ФОРТРАНе, автоматическая генерация блок-схем программ, обработка платежных или налоговых ведомостей).

В разд. 6.2 мы рассмотрим некоторые фундаментальные технические аспекты проблемы выбора. После того как в разд. 6.2.1 читателю будет дан основательный, но краткий обзор методики конкурсного выбора, мы обсудим в разд. 6.2.2 вес оценки производительности в выборе решения и главные препятствия, с которыми сталкивается оценщик, приступая к проблемам выбора. В разд. 6.3 мы будем интересоваться сравнением производительностей с целью выбора. После обсуждения некоторых общих проблем сравнения в разд. 6.3.1 мы рассмотрим более подробно сравнение способов обработки в разд. 6.3.2. Раздел 6.3.3 посвящен проблеме выбора поставщика, разд. 6.3.4 — выбору установки, а разд. 6.3.5 — выбору компонентов системы. Проблемы выбора, которые мы отнесли к классу d, рассмотрены в разд. 9.4.1. В разд. 6.4 проблемы выбора трактуются с точки зрения методики. Некоторые из специфических и крайне сложных вопросов описания рабочей нагрузки, которые ставит задача выбора, рассмотрены и обсуждены в разд. 6.5.

6.2. МЕТОДЫ ВЫБОРА

6.2.1. Процедура конкурсного выбора

Выбор компьютера выдвигает, как это станет очевидно из продолжения данной главы, крайне трудноразрешимые вопросы. Не существует универсальной, стандартизированной процедуры получения ответа на эти вопросы. В лучшем случае стандартные правила выбора существуют лишь внутри организации. Эти правила, однако, зачастую существенно отличаются от правил, принятых в других организациях. Многие организации также пользуются нестандартизированными подходами ad hoc. В этом разделе мы обрисуем в общих чертах возможную процедуру решения различных классов проблем выбора, представленных в разд. 6.1. Нашей целью является обоснование роли оценки производительности в этих проблемах. При изложении этой процедуры мы будем следовать Тиммреку (1973), который применяет ее, в частности, для конкурсного выбора поставщика. Процедура состоит из четырех шагов.

Шаг 1. Анализ и точное определение потребностей. Это решающий и, возможно, самый трудный шаг. Он включает в себя такие проблемы, как описание рабочей нагрузки, прогнозирование рабочей нагрузки, анализ потоков информации, анализ и прогнозирование запросов пользователей. Если желательна процедура беспристрастного выбора, потребности должны быть сформулированы в виде, независимом от поставщика и системы. Мы уже видели в разд. 5.2.3, в чем состоят главные препятствия для действительно системно-независимого описания рабочей нагрузки. Большая часть тех же доводов может быть повторена для остальных перечисленных выше проблем, относящихся целиком к определению окружающей среды системы. На этом шаге должно быть подготовлено (если возможно, количественное) описание окружающей среды и вероятного развития. Ясно, что эта задача легче в случае, когда предстоит выбрать новую систему для существующей установки, нежели в случае, когда новую установку еще предстоит смонтировать. Те аспекты этого шага, которые теснее связаны с оценкой производительности, обсуждаются в разд. 6.5.

Шаг 2. Подготовка заявок. Потребности, идентифицированные на шаге 1, должны быть преобразованы в ряд требований, предъявляемых различным поставщикам. Поставщики, по всей вероятности, предложат конфигурацию, отвечающую этим требованиям. Разумеется, преобразование не должно менять требования в пользу отдельных поставщиков. Полезно различать *обязательные* и *желательные* требования и минимизировать количество обязательных [см. Джослин (1968)]. Требования будут включать спецификации производительности, характеристики оборудования, режимы обработки, свойства операционной системы, стоимость системы, спецификации надежности и легкости эксплуатации системы, потребности в обслуживании и требования к квалификации персонала, языки и пакеты прикладных программ и описания возможностей расширения и замены.

Шаг 3. Проверка пригодности предложенной системы. Когда получены предложения поставщиков, их следует проверить на приемлемость. При этом выясняется, удовлетворяет ли и в какой степени каждая из предлагаемых систем выдвинутым требованиям. Проверки выполняются с помощью справочников, приемов и средств оценки, результатов измерений, представленных поставщиками, и информации с других установок, оснащенных той же системой. Предложения, не отвечающие всем обязательным требованиям, отвергаются. Остальные предстоит сравнить на шаге 4.

Шаг 4. Отбор среди предложений, прошедших проверку. Сравнение предложений, прошедших проверку на шаге 3, может быть выполнено различными способами. *Методы ad hoc* состоят обычно из простых сравнений характеристик и оценок, показывающих, насколько эти предложения удовлетворяют желаемым требованиям. Такие методы часто используются, когда процедура отбора должна показать, есть ли основания выбирать иного поставщика вместо предпочитаемого данным потребителем. *Методы взвешивания* [см., например, Шарп (1969)] требуют, чтобы было подготовлено дерево характеристик системы посредством последовательных делений характеристик на классы и подклассы, пока не будут включены все подходящие характеристики. Эти характеристики могут совпадать с требованиями, специфицированными в шагах 1 и 2. Затем каждой характеристике приписывается численный вес, который должен представлять ее относительную важность для оценщиков. Сумма весов, приписанных всем подузлам любого данного узла, должна быть равна 1. Наконец, для каждого предложения приписывают оценку каждому листу дерева и эти оценки умножают на соответствующие веса. Эти произведения для узлов, имеющих общий порождающий узел, затем складываются и дают оценку этого узла [1]. Этот процесс повторяется до тех пор, пока не будет получена оценка каждой системы. Выбирается предложение с наибольшей оценкой. Основные классы характеристик, использованные в исследовании, проводимом в корпорации RAND [см. Шарп (1969), стр. 285], основывались на методе взвешивания. Они приведены в табл. 6.1. *Метод цены и ценности*, предложенный Джослином (1968), приписывает денежную стоимость (ценность) желаемым требованиям, перечисленным в шаге 2. Эта стоимость вычитается из цены тех предложений, которые удовлетворяют соответствующим желаемым требованиям. Выбирается предложение, окончательная цена которого минимальна. Читатель уже наверняка заметил, что субъективное взвешивание играет не слишком малую роль во всех описанных выше методах. Оценка преимуществ и недостатков этих методов выходит за рамки этой книги и может быть найдена в работе Тиммрека (1973).

Необходимо сделать два важных замечания относительно описанной выше процедуры. Первое состоит в том, что многие выборы выполняются в действительности посредством так называемой *процедуры единственного источника*. При выборе поставщика это означает, что поставщик известен заранее, а проблема состоит в разработке конфигурации. Это более ограниченная, но нетривиальная проблема, особенно если желательно рассмотреть покупку или аренду компонентов аппаратуры или программного обеспечения от независимых изготовителей (см. гл. 8). Закупки из «единственного источника» могут быть следствием разных причин. Наиболее важная из них — недопустимо высокая цена преобразования программ, которого могла бы потребовать система другого поставщика. Это затруднение иногда успешно решается с помощью эмуляции машины, которую надо заменить. Однако есть и другие причины, по которым новая система приобретается у того же поставщика, что и старая: удовлетворение полученным обслуживанием; привлекательные статьи договора, скидки и будущие выгоды; более легкое обучение программистов и обслуживающего персонала; экономические преимущества. В определенных ситуациях превосходство нового поставщика в некоторых из этих областей побуждает сменить поставщика.

Второе замечание состоит в том, что процедуру выбора, подобную процедуре, проиллюстрированной выше, разумно использовать только тогда, когда цена системы, которую предстоит выбрать, оправдывает необходимые усилия. Более простые и дешевые процедуры выбора следует применять, скажем, для выбора отдельного мини-компьютера.

Процедуры для решения проблем выбора установок или компонентов системы могут концептуально копировать только что описанные. На практике, поскольку количество денег, отпущенных на этот выбор, в общем случае меньше общей стоимости средней или большой системы, а возможные исходные альтернативы более малочисленны и лучше специфицированы, то процедуры, которым в действительности следуют, короче, проще и дешевле. Обычно от возможных поставщиков не ожидают предложений, а просто они предоставляют техническую информацию и информацию о ценах тех изделий, которые могли бы удовлетворить потребности пользователей. Оценивая эти данные и сравнивая их с установленными потребностями, пользователь может прийти к решению. Например, можно подойти к проблеме выбора установки посредством следующих шагов:

a) определение рабочей нагрузки, подлежащей обработке (шаг 1);
b) сбор информации о производительности и стоимости доступных установок (шаг 2);
c) оценка собранной информации и отказ от установок, не выдерживающих конкуренции с остальными (шаг 3);
d) сравнение цен и производительностей остальных установок с целью найти наиболее подходящую для данной задачи (шаг 4).

Некоторые из приемов, которые можно применить для выполнения этих операций, описаны в разд. 6.3.4. Выбор компонента системы также можно легко описать как следование той же самой последователь-

[1] В данной терминологии каждый узел соответствует классу, а его подузлы — подклассам этого класса. Лист — это узел, не порождающий других узлов (иначе говоря, класс, не разделяемый на подклассы).— *Прим. ред.*

ности шагов. Следует, однако, отметить, что неформальные конкурсные процедуры или покупка «из единственного источника» даже более широко используются при выборе установок и компонентов системы, чем при выборе поставщика.

6.2.2. Роль оценки производительности в процедурах выбора

В разд. 6.2.1 мы вкратце описали шаги процедуры решения проблем выбора. Эта процедура специально приспособлена к проблемам выбора поставщика. Однако та же самая процедура с некоторыми упрощениями, как выяснилось, концептуально подходит также для выбора установки и компонентов системы. Даже поверхностное исследование этой процедуры показывает, что она включает различные действия и соображения, строго говоря, не относящиеся к области оценки производительности, как мы ее определили в разд. 1.4. Экономические аспекты вполне естественно присутствуют во всех шагах процедуры. Должны быть приняты во внимание некоторые, по существу качественные характеристики, такие, как особенности языка и средства программирования. Придание им количественного смысла (при такой попытке, как в методе взвешивания) всегда есть выражение субъективных мнений, к которым не применима методология оценки, обсуждаемая в этой книге. В этом разделе мы исследуем роль, которую играет количественная оценка производительности в процедуре выбора. В каких местах процедуры, описанной в разд. 6.2.1, необходимо или удобно производить оценки того типа, который рассматривается в этой книге (см. разд. 1.4), и каков их вклад в процесс выбора?

Некоторые из потребностей, которые нужно проанализировать и сформулировать на шаге 1 этой процедуры, можно обозначить термином *требования производительности*. Пользователи проектируемой установки или компонента системы, который предстоит приобрести, или создаваемой вычислительной службы в общем случае предъявляют требования трех типов: функциональные требования, требования производительности и экономические требования. Они хотят, чтобы их потребности в обработке информации были удовлетворены на определенном уровне производительности и за минимальную цену. Эти три типа требований взаимосвязаны, так как каждое из рассматриваемых свойств системы (или компонента) имеет функциональные аспекты, связанные с производительностью и со стоимостью. Различие между обязательными и желательными требованиями, упомянутое в разд. 6.2.1, относится к каждому типу требований. Это различие важно тем, что предоставляет принимающему решение свободу выбора компромисса между функциональными возможностями и аспектами производительности, с одной стороны, и их ценами, с другой. Анализ и определение требований производительности включает в себя решение относительно основных индексов производительности и критериев (см. разд. 1.4), посредством которых можно оценивать и сравнивать различные варианты, и выбор для них численных значений или диапа-

зонов. Так как производительность системы зависит от ее рабочей нагрузки, рабочая нагрузка тоже должна быть охарактеризована и описана. Таким образом, если проектируется установка, следует использовать методы описания и прогнозирования рабочей нагрузки и моделирования коллектива пользователей для анализа и конкретизации запросов (шаг 1). В случае выбора установки рабочая нагрузка, вообще говоря, известна только частично. Предсказать, какие программы будут делить ресурсы системы с программами пользователя, обычно невозможно. Однако эта дополнительная рабочая нагрузка должна быть оценена, если нужно определить производительность, которую система будет обеспечивать для пользователя. При выборе компонента подробное описание его входной информации не обязательно. Технические характеристики, не зависящие от входной информации, могут оказаться достаточными для спецификации производительности компонентов, особенно оборудования. Например, производительность модуля основной памяти можно удовлетворительно описать длиной слова, временем доступа, временем записи и скоростью передачи данных, а производительность дисковода — его характеристиками поиска, временем оборота, скоростью передачи, организацией данных. Во всех задачах выбора шаг 1 крайне важен. Пользователям следует иметь ясное представление о производительности (и цене реализации) запрошенных ими компонентов и конфигурации систем. В частности, легко представить себе противоречивые или нереальные спецификации, которых, очевидно, следует избегать. Таким образом, некоторые работы по оценке могут оказаться необходимыми или по крайней мере полезными на шаге 1, чтобы помочь конкретизировать запросы. Целью этих работ могло бы быть определение влияния особенностей системы на данные индексы производительности, проверка адекватности некоторых индексов или свойств предполагаемой рабочей нагрузки.

Подготовка заявок (шаг 2) может потребовать тех же типов исследования, что и анализ потребностей. На практике, однако, работы по оценке редко выполняются на протяжении первых двух шагов, несмотря на то что они часто были бы весьма полезны. Исследования, которые требуются на этих двух шагах, не обладают особенными характеристиками, присутствующими только в задачах выбора. Выбор индексов производительности, их анализ, описание и прогнозирование рабочей нагрузки используются на этапах улучшения производительности или проектирования систем.

Намного более типичные характеристики выявляются в работах по оценке, необходимых для обоснования (шаг 3) и окончательного выбора (шаг 4) заявок. Шаг 3 является единственным, на котором методы оценки производительности играют существенную роль, по крайней мере в ряде случаев практического выбора. Разумеется, мы полагаем, что эта роль должна быть расширена, чтобы включить также шаги 1, 2 и 4. Но мы не можем игнорировать тот факт, что современные методы оценки почти исключительно используются (если используются вообще) для обоснования различных заявок на шаге 3. В большинстве

случаев остальные шаги следуют субъективным методам, даже если существуют более объективные количественные подходы.

Обоснование заявки с точки зрения производительности является типичной задачей анализа производительности (см. разд. 1.4). Если задан набор спецификаций, определенных на шагах 1 и 2, цель состоит в том, чтобы увидеть, удовлетворены или нет обязательные требования и насколько хорошо удовлетворены желательные требования.

Таблица 6.1

Простая иллюстрация методов взвешивания

Характеристика системы	Оценки (в диапазоне 0—10)			
	Вес*	Поставщик А	Поставщик В	Поставщик С
Аппаратура	0.27	7	8	6
Операционная система	0.27	5	6	7
Управление данными	0.08	6	4	4
Языковые процессоры	0.16	5	7	8
Общие средства программирования	0.02	7	5	7
Возможности переноса на другую систему	0.12	9	6	7
Надежность поставщика и обеспечиваемое им сопровождение	0.08	6	7	5
	1.00			
Взвешенные оценки		9.46	10.38	10.0

* *Источник*: Шарп (1969).

Однако в шаге 3 подразумевается *сравнение производительности* различных вариантов, что придает ему особую специфику. Проблема сравнения, которая рассматривается в шаге 4 с глобальной точки зрения (точка зрения, включающая функциональные характеристики, характеристики производительности и стоимости) не является особенностью процедур выбора. Даже проблемы проектирования и усовершенствования можно свести к сравнению вариантов. Что действительно является особенностью проблем выбора, так это то, что сравнивать приходится варианты, обычно намного больше отличающиеся друг от друга, чем в других типах задач.

Таким образом, обоснования, которые необходимо получить на шаге 3, должны быть нацелены на обеспечение исследователя не только доводами для решения продолжать ли перебор вариантов или остановиться, но и количественной информацией об относительных производительностях, которые, по-видимому, будут показаны различными вариантами в заданной среде.

Какое воздействие оказывают результаты сравнения производительностей на решение о выборе, которое придется принять на шаге 4? Методы, описанные в разд. 6.2.1, не делают четкого различия между функциональными характеристиками, характеристиками производительности и стоимости. Как в методе взвешивания, так и в методе «цена — ценность» аспекты производительности перемешаны с аспектами другого рода. Точнее, методы взвешивания обычно пытаются выразить количественно ценность каждого из конкурирующих решений для будущих пользователей. Оценка, приписанная предложению, включает в себе его функциональные аспекты и аспекты производительности, тогда как его цена обычно приводится отдельно. Несколько попыток учета цены в весе оказались неудачными из-за трудности приписывания цене веса, согласованного с весами других характеристик системы. Таким образом, решение о выборе, основанное на методе взвешивания, заключает явное предположение о компромиссе между ценой и ценностью. Заметим, что отделение цены от весов помогает нам прояснить варианты, но не решает проблему выбора. Те же самые трудности, которые возникают при попытке приписать вес цене, проявляются на этапе принятия решения. Например, пусть предложенные в табл. 6.1 варианты имеют цены 1 550 000, 1 870 000 и 1 750 000 долларов соответственно. Какой выбрать? Заметим также, что если окончательные оценки так близки друг к другу, как в табл. 6.1, вполне обоснованно подозрение, что их различия в большой степени обусловлены «шумом» в системе весов. Различия, настолько малые, как в табл. 6.1, следует, вероятно, рассматривать как несущественные.

Метод «цена — ценность», выражая все характеристики системы в денежных единицах, позволяет при принятии решения не встречаться прямо с компромиссом «цена — ценность». Однако с этим компромиссом приходится иметь дело фактически на каждом шаге метода, скорее на микроскопической основе (элемент за элементом), чем на макроскопической основе (в конце работы). Для каждой желаемой потребности системы мы должны определить цену обладания ею, цену отказа от нее, цены, связанные со сроками удовлетворения этой потребности, и т. д.

Таким образом, компромисс между ценой и ценностью является решающим в обоих методах выбора. А как насчет компромисса «стоимость — производительность»? Важно понять, что недопустимо строить выбор, только исходя из соображений цены и производительности. Такие функциональные аспекты, как сопровождение системы, наличие программ обслуживания преобразований, обучение, возможности расширения, легкость использования, доступное программное обеспечение (в частности, языки программирования), документация, совместимость с существующей системой, оказывают (и по праву) большое влияние на решения о выборе. Производительность в том ограниченном смысле, в каком этот термин используется в этой книге, также является важным фактором, даже несмотря на то что в некоторых случаях все еще используются примитивные индексы (например, в

проблеме выбора поставщика это — время цикла памяти, время сложения с фиксированной точкой, максимальный размер памяти) вместо индексов, более прямо связанных с тем, насколько хорошо работает данная система.

Вне зависимости от того, какой метод выбора используется на шаге 4, необходимо сравнивать системы по степени удовлетворения потребностей. В подходе со взвешиванием должно быть проведено сравнение различных вариантов для каждой характеристики системы, чтобы назначить оценки. В методе «цена — ценность» сравнения не явно присутствуют в приписывании цен различным предложениям. Таким образом, сравнение производительностей — существенный компонент процедуры выбора, даже если действительное решение основывается только на совокупности результатов выбора и компромисса «стоимость — производительность», а не на каком-либо одном из этих факторов. Сравнение производительностей будет основной темой разд. 6.3.

6.3. СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТЕЙ

6.3.1. Задача сравнения

В этом разделе мы рассмотрим с концептуальной точки зрения следующий вопрос: насколько надежен результат работы по сравнению производительностей?

Рассмотрим два варианта A и B, производительности которых нужно сравнить. Целью сравнения является выбор лучшего варианта и придание этому выбору по возможности количественного вида (т. е. возможности сказать, например, что B лучше A на 20%). Предположим, что выбрано множество $\mathcal{P}$ индексов производительности и что оно адекватно представляет наше понятие о производительности для A и B. Выбор $\mathcal{P}$ является важной и трудной проблемой, которая будет обсуждаться в следующих разделах, а здесь мы будем рассматривать ее как уже удовлетворительно решенную. Мы также предположим, что из индексов в $\mathcal{P}$ выведен удобный критерий F, так что сравнение величин F для двух вариантов показывает, какой из них лучше и насколько. Это, очевидно, еще одна нетривиальная проблема, относящаяся к последующим разделам. Заметим, что если в $\mathcal{P}$ существует только один индекс или индекс преобладающей важности, этот индекс и будет взят в качестве критерия. Кроме того, исследователя может интересовать сравнение производительности «индекс за индексом». В этом случае для каждого сравнения F будет очередным индексом из $\mathcal{P}$.

Таким образом, при идеальных условиях сравнение F_A и F_B даст нам желаемый ответ. На практике, однако, существует несколько источников ошибок, которые могут исказить значения F_A и F_B и сделать их сравнение ненадежным. При предположениях, сделанных выше, двумя главными источниками ошибок могут быть различные ус-

вия окружения, в которых определены F_A и F_B, и статистическая природа по крайней мере некоторых индексов, от которых зависит F.

Первый источник обусловлен наличием в $\mathcal{P}$ индексов, значения которых зависят не только от производительностей A и B, но также и от среды, в которой действуют A и B. Это имеет место практически для всех наиболее широко распространенных индексов производительности вычислительных систем, значения которых критически зависят от рабочей нагрузки. Чтобы сделать их описателями производительности A и B, мы должны задать одну или более рабочих нагрузок и определить их значения при этих нагрузках как для A, так и для B. Только несколько индексов могут считаться нечувствительными к рабочей нагрузке. Среди них оказываются определенные параметры компонентов, например времена доступа к памяти, максимальные скорости передачи информации, времена сложения, верхние или нижние границы индексов производительности системы (т. е. емкость или максимальная пропускная способность, максимальное количество терминалов, обслуживаемых диалоговой системой в режиме редактирования при заданном среднем времени ответа). Однако легко видеть, что даже определения этих границ требуют тщательного задания учитываемых типов входных воздействий. Таким образом, все факторы, влияющие на F, кроме тех, которые характеризуют два варианта A и B, должны контролироваться так, чтобы уничтожать их возможные вклады в разность $F_A - F_B$. Эта разность должна отражать только относительные производительности A и B. К сожалению, на практике весьма трудно достичь этой цели. Например, очень трудно охарактеризовать рабочую нагрузку установки таким образом, чтобы можно было сказать, что две системы A и B управляются одинаковой рабочей нагрузкой (эта проблема подробно обсуждается в разд. 5.2.3 и 6.5). Выбор этой рабочей нагрузки — еще один важный, но нетривиальный вопрос (см. разд. 5.2.2 и 6.3.3). В принципе индексы и рабочая нагрузка (или рабочие нагрузки), выбираемые для сравнительного исследования, должны по возможности подчеркивать различия в производительности вариантов, одновременно минимизируя влияние возможных отличий рабочих нагрузок. Заметим, что представительность модели рабочей нагрузки — не очень строгое требование, поскольку низкая степень представительности не искажает существенно $F_A - F_B$.

Другим источником ошибок, который будет вкратце обсужден в этом разделе, является статистическая природа большинства индексов производительности, от которых зависит критерий F. Исследователи обычно интересуются численными значениями характеристик выборки, например средними значениями этих индексов, но из практически собираемых конечных наборов наблюдений можно получить лишь оценки этих описателей (т. е. выборочные средние). Таким образом, можно вывести неверные заключения из сравнения производительностей, в котором индексы не были оценены точно. Пусть F — индекс производительности системы, например среднее время $\bar{t}$ прохож-

дения задания. Мы будем считать, что система A имеет более высокую производительность, чем система B по отношению к $\bar{t}$ при определенной рабочей нагрузке, если $\bar{t}_A < \bar{t}_B$. В идеале рабочая нагрузка, используемая в этом сравнении, должна быть такова, что она могла бы обрабатываться любой из этих систем, как только система выбрана и установлена. Поскольку, как объяснено в разд. 5.1, экспериментировать с такой рабочей нагрузкой практически невозможно, эти два средних времени прохождения должны оцениваться по выборкам, полученным от систем, управляемых моделями этой рабочей нагрузки. Если $\hat{\bar{t}}_A$ и $\hat{\bar{t}}_B$ — оценки $\bar{t}_A$ и $\bar{t}_B$ по этим выборкам, их сравнение должно

a) дать одинаковые качественные результаты: если $\bar{t}_A > \bar{t}_B$, то $\hat{\bar{t}}_A > \hat{\bar{t}}_B$; если $\bar{t}_A < \bar{t}_B$, то $\hat{\bar{t}}_A < \hat{\bar{t}}_B$; если $\bar{t}_A \approx \bar{t}_B$, то $\hat{\bar{t}} \approx \hat{\bar{t}}_B$;

b) дать сходные количественные результаты: оценка процента превышения производительности одной системы над другой не должна слишком отличаться от действительного процента превышения.

Следует отметить, что мы в действительности оцениваем разность $d_{AB} = \bar{t}_A - \bar{t}_B$ по разности $\hat{d}_{AB} = \hat{\bar{t}}_A - \hat{\bar{t}}_B$ между соответствующими выборочными средними. Поэтому наша задача может быть сформулирована в следующем виде: вычислить доверительный интервал для d_{AB} при заданном доверительном уровне K или иначе вычислить вероятность попадания значения d_{AB} вне заданного доверительного интервала около $\hat{d}_{AB}$. Мы можем (по крайней мере в принципе) экспериментировать с рядом различных выборок из реальной рабочей нагрузки. Каждая выборка даст нам две оценки среднего времени прохождения, одну для системы A и одну для системы B. Следовательно, можно определить распределения $\hat{\bar{t}}_A$ и $\hat{\bar{t}}_B$. Если $\hat{\bar{t}}_A$ и $\hat{\bar{t}}_B$ независимы и распределены по нормальному закону, их разность также распределена нормально. Таким образом, доверительный интервал для d_{AB} задается выражением

$$\hat{d}_{AB} - z\sigma_{AB} \leqslant d_{AB} \leqslant \hat{d}_{AB} + z\sigma_{AB}, \tag{6.1}$$

где z определяется по (2.5), а σ_{AB} — стандартное отклонение $\hat{d}_{AB}$ относительно d_{AB}. Если известны дисперсии σ_A^2 и σ_B^2 величин $\hat{\bar{t}}_A$ и $\hat{\bar{t}}_B$, предположение о независимости позволит нам найти σ_{AB}^2 как сумму $\sigma_A^2 + \sigma_B^2$. Однако это обычно не так. Поэтому дисперсии должны оцениваться по формуле [см. (2.7) и (2.13)]

$$\hat{\sigma}_A^2 = \frac{\sum_{i=1}^{n_A} (t_{Ai} - \hat{\bar{t}}_A)^2}{n_A (n_A - 1)} \tag{6.2}$$

и по аналогичной формуле для $\hat{\sigma}_B^2$. Пусть $n_A = n_B = n$ и $\hat{\sigma}_{AB}^2 = \hat{\sigma}_A^2 + \hat{\sigma}_B^2$. Так как случайная переменная $(\hat{d}_{AB} - d_{AB})/\hat{\sigma}_{AB}$ подчинена распре-

делению Стьюдента (t-распределение), то вычисление доверительного интервала в (6.1) должно основываться скорее на нем (особенно если n не очень велико), нежели на нормальном распределении:

$$\hat{d}_{AB} - z'\hat{\sigma}_{AB} \leqslant d_{AB} \leqslant \hat{d}_{AB} + z'\hat{\sigma}_{AB}, \tag{6.3}$$

где z' связано с K, как показано в разд. 2.4.3, и может быть найдено в таблице t-распределения с $n-1$ степенями свободы.

Сделанные нами предположения существенны. Если, например, $n_A \neq n_B$, то t-распределение Стьюдента нельзя использовать, если не выполнено $\sigma_A = \sigma_B$ [см. Снедекор и Кокрен (1967), стр. 115, для полезного приближения]. Предположения о нормальности и независимости устранить труднее. Соображения, аналогичные сделанным в разд. 2.6.4 для вычисления минимального размера выборки без предположения о нормальности и независимости, можно применить с необходимыми модификациями к вычислению доверительных интервалов для d_{AB}. Например, если можно лишь предположить, что $\hat{d}_{AB}$ имеет конечное среднее и конечную дисперсию, неравенство Чебышева [использованное при выводе (2.10)] дает

$$\hat{d}_{AB} - \sigma_{AB}/\sqrt{1-K} \leqslant d_{AB} \leqslant \hat{d}_{AB} + \sigma_{AB}/\sqrt{1-K}. \tag{6.4}$$

Если выполнение предположения о нормальности маловероятно, можно использовать для этой же цели непараметрические методы, например ранговые тесты. Описания этих методов можно найти в большинстве справочников по статистическому анализу. Однако заметим, что распределение $\hat{d}_{AB}$ ближе к нормальному, чем распределения $\hat{\bar{t}}_A$ и $\hat{\bar{t}}_B$, если распределения $\hat{\bar{t}}_A$ и $\hat{\bar{t}}_B$ обладают асимметрией одного знака.

Когда доверительный интервал для d_{AB} оценен, сравнение его ширины со значением $\hat{d}_{AB}$ дает нам полезную информацию о возможном знаке d_{AB} и о его значимости. Если доверительный интервал, соответствующий доверительному уровню K, включает нуль, d_{AB} имеет заметную вероятность оказаться по любую сторону начала координат (в котором обе сравниваемые системы обладают одинаковой производительностью). Следовательно, мы не сможем отвергнуть гипотезу $d_{AB} = 0$ при данном уровне значимости $1-K$, и разность между этими двумя производительностями статистически незначима. Понижение уровня значимости сделает нашу оценку более точной, а наши тесты более чувствительными, но потребует много большего размера выборки (см. разд. 2.4.3 и 2.6.4).

Если модели рабочей нагрузки, использованные для A и B, не только представляют одну и ту же рабочую нагрузку (что является существенным требованием, налагаемым соображениями, выдвинутыми в начале этого раздела), но еще и состоят из того же числа моделей заданий, мы можем объединить в пару наблюдения t_{Ai} и t_{Bi}, соответствующие заданию i. Дисперсия $\hat{d}_{AB}$ теперь уже не равна сумме дисперсий $\hat{\bar{t}}_A$ и $\hat{\bar{t}}_B$, как в случае независимости, поскольку ковариация

между выборочными средними, вероятно, не мала:

$$\sigma^2_{AB} = \sigma^2_{AB} + \sigma^2_B - 2\mathrm{Cov}\,(\hat{\bar{t}}_A,\ \hat{\bar{t}}_B). \tag{6.5}$$

Объединение в пары сделает $\mathrm{Cov}\,(\hat{\bar{t}}_A,\ \hat{\bar{t}}_B)$ положительной, тем самым уменьшая σ_{AB} и увеличивая точность сравнения для выборок одинаковой длины по сравнению с случаем независимых наблюдений. Предположение о независимости в случае наблюдений, фактически объединенных в пары, может привести к большей ширине доверительного интервала и к пренебрежению величиной d_{AB}, тогда как она на самом деле значима. В предположении, что разности $d_{ABi}-d_{AB}$, где $d_{ABi}=t_{Ai}-t_{Bi}$, независимы и распределены нормально с нулевым средним и дисперсией σ^2_d, значения $\hat{d}_{AB}$ распределены нормально около d_{AB} с дисперсией σ^2_d/n, а σ^2_d можно оценить по выборке с помощью формулы

$$\hat{\sigma}^2_d = \frac{\sum_{i=1}^{n} (d_{ABi} - \hat{d}_{AB})^2}{n-1}, \tag{6.6}$$

Так как переменная $(\hat{d}_{AB}-d_{AB})/(\hat{\sigma}_d/\sqrt{n})$ подчиняется распределению Стьюдента, то доверительный интервал для d_{AB} при доверительном уровне K дается формулой (6.3), где $\hat{\sigma}_{AB}$ следует заменить на $\hat{\sigma}_d/\sqrt{n}$.

Другими важными и иногда упускаемыми из внимания источниками ошибок в сравнении производительностей являются различные смещения, внесенные в результат погрешностями в использовании методов оценки за счет, например, неточных измерителей, неточных имитационных или аналитических моделей и неточной записи данных или неправильного манипулирования ими. Следует заметить, что любая систематическая ошибка, в одинаковой мере влияющая на величины индексов производительности обеих систем, не оказывает нежелательного воздействия на результат сравнения. Таким образом, следует уделять гораздо большее внимание проблеме повышения однородности входной информации, методов оценки, измерителей и процедур для сравниваемых систем, чем получению точных абсолютных оценок для индексов производительности каждой системы. Читателю следует также учесть, что совокупный вклад описанных в этом разделе источников дисперсии в $\hat{d}_{AB}$, а иногда (при должным образом спланированных экспериментах) и даже вклад каждого из источников можно оценить методом дисперсионного анализа (см. разд. 2.7.2).

6.3.2. Сравнение режимов обработки

В некоторых работах по проектированию установок одним из первых приходится принимать решение о выборе режима обработки. Это решение оказывает очень важное влияние на выбор поставщика, на проект конфигурации аппаратуры и программного обеспечения

и на организацию установки. Однако проблема выбора режима обработки редко обсуждается и решается формально. Основная причина этого состоит в том, что каждый режим обработки имеет своеобразные и уникальные характеристики, делающие его удобным для одних типов применения и непригодным для других. Поэтому, если задано применение, выбор режима обработки для него обычно ясен. *Прямой доступ* очень уместен на установках с единственным пользователем или на очень малых установках с доступом пользователя к машине, ориентированных на удобство работы для ограниченного числа сравнительно квалифицированных пользователей. *Пакетный режим* удобен для установок среднего размера, рабочая нагрузка которых состоит в основном из производственных программ, требущих незаурядной вычислительной мощности или мощности ввода-вывода или и того, и другого. Для установок, где преобладающим родом деятельности является разработка программ, обычно предпочтителен *пакетный режим с быстрым прохождением заданий* или диалоговый. Последний способ также наиболее подходит для таких приложений, как проектирование с помощью компьютера, машинная графика, информационный поиск, подготовка документов, системы предварительных заказов. Режим работы *в реальном времени*, в некотором смысле подобный диалоговому режиму, обычно используется для управления промышленными процессами или лабораторными экспериментами.

Другая причина того, что выбору режима обработки редко уделяется серьезное внимание, состоит в том, что большие системы, обычно устанавливаемые в случае, когда требуется разнообразие применений, часто снабжаются операционными системами, обеспечивающими несколько режимов обработки одновременно (обычно пакетный и диалоговый режимы, но иногда также и режим реального времени).

Таким образом, проблема выбора режима обработки лишь изредка рассматривается в процедуре выбора, поскольку очень часто решается непосредственно. Тем не менее в этой области было проведено несколько сравнительных исследований, а их результаты должны бы заинтересовать как разработчиков установок, так и проектировщиков систем. Более того, некоторые методологические аспекты этих исследований интересны и применимы для целого ряда проблем, включая выбор режима обработки (когда имеется либо операционная система с многими режимами обработки, либо многомашинный комплекс с различными режимами обработки), проектирование и оценку средств связи пользователя с машиной, измерение и моделирование поведения коллектива пользователей. Поэтому мы обсудим вкратце сравнение двух режимов обработки, пакетного и диалогового, используя в качестве главного источника информации обзор Сакмана (1968) некоторых ранних экспериментальных работ в этой области.

Первый вопрос, на который придется ответить, относится к целям исследования: что означает сравнение пакетного и диалогового режимов? Разумеется, по этому вопросу существует несколько различных точек зрения и все они законны и интересны. Если мы примем точку

зрения проектировщика установки, главная цель которого состоит в максимизации ценности установки для данной организации, мы столкнемся с вопросом, каково влияние режима обработки на ценность установки. Общий ответ на этот вопрос дать невозможно. Влияние зависит от типа и применения установки, например от доли работ по развитию программ и доли производственного счета, от способностей и подготовки пользователей, от того, как управляется установка, и т. д. Если необходимо получить результаты, в которых заинтересованы многие люди, а не только администраторы данной установки, следует отказаться от идеи определения и использования ценности как критерия сравнения и предпочесть выбор нескольких количественных индексов, которым любой заинтересованный человек мог бы затем придать различные веса в процессе принятия частного решения. Эти индексы должны придать некоторую количественную форму следующим важным аспектам нашего сравнения:

a) расход человеческих ресурсов,
b) расход системных ресурсов,
c) количество работы, производимой программистами,
d) качество работы, производимой программистами,
e) отношение пользователей.

Заметим, что производительность системы не включена в этот список классов индексов, так как аспекты производительности системы

Таблица 6.1

Индексы для сравнения пакетного и диалогового режимов обработки

Класс	Индекс
Затраты человеческих ресурсов	Время программиста
Затраты системных ресурсов	Машинное время
Количество проделанной работы	Затраченное время
Качество проделанной работы	Время выполнения программы, размер программы
Отношение пользователей	Предпочитаемый режим

представлены индексами расхода системных ресурсов, а аспекты реактивности системы, как ожидается, серьезно влияют на остальные четыре класса индексов. В табл. 6.2 приведены некоторые индексы, принадлежащие этим классам, которые в действительности использовались в упомянутых выше сравнительных исследованиях. Эти индексы были измерены во время выполнения специфического программного задания рядом программистов с использованием этих двух сравниваемых режимов.

В идеальном случае эксперимент по сравнению должен быть спланирован так, чтобы исключить все источники изменения этих индексов, кроме режима обработки. Таким образом, одной и той же группе программистов следует дать одну и ту же задачу для выполнения в обоих режимах. Эти два режима должны обеспечиваться одной системой или очень похожими системами с соизмеримыми по величине нагрузками. Задание на программирование не должно быть ориентировано ни на пакетную, ни на диалоговую обработку. Тем не менее следует адекватно представить применения, характерные и существенные для обоих типов установок. Ясно, что большая часть этих требований может быть удовлетворена в реальном эксперименте лишь приблизительно и должны использоваться методы управления вторичными факторами (см. разд. 2.6.2 и 2.6.3). При планировании эксперимента придется ответить на несколько сложных вопросов. Например, сколько потребуется программистов и насколько они должны быть квалифицированными? Если они новички, результаты могут оказаться бесполезными для установок, на которых большую часть коллектива пользователей составляют высококвалифицированные программисты. Если у них есть некоторый предварительный опыт, они, возможно, будут склоняться к режиму, в котором они работали с машиной в прошлом. Кроме того, поскольку невозможно заставить одних и тех же программистов решать данную задачу в обоих режимах, эти две группы следует тщательно подобрать, а для получения результатов можно воспользоваться методом объединения в пары. Другие вопросы относятся к выбору программных заданий и используемого языка. Чтобы устранить изменчивость, обусловленную различием языков, в обоих режимах должен использоваться один язык. На этом месте можно выдвинуть возражение, что не реально не давать программистам использовать языки, специально предназначенные для работы в каждом из этих режимов обработки. Однако в нашем сравнении особое значение придается именно индексам в табл. 6.2, относящимся к различным характеристикам времени ответа при этих двух режимах обработки, а не к относительным преимуществам конкретной пакетной и конкретной диалоговой системы. Это замечание полезно при рассмотрении следующего вопроса: следует ли управлять производительностью этих двух систем во время эксперимента и как? Ясно, что, даже если для этих двух режимов используется одна и та же конфигурация оборудования, различие рабочих нагрузок на систему в период, когда программисты, участвующие в эксперименте, тестируют и отлаживают свои программы, обусловит изменчивость характеристик ответа. В пяти работах, рассмотренных в обзоре Сакмана (1968), приняты два противоположных подхода. Подход отказа от управления, которого придерживаются в четырех из этих исследований, основан на предположении, что управление производительностью было бы не реалистичным. Единственная работа, в которой использовалось управление нагрузкой,— работа Гранта и Сакмана (1966), где время прохождения в пакетной системе поддерживалось постоянным и равным 2 часам. Вероятно, лучшим решением было бы управлять временем прохождения так, чтобы оно колебалось в обычных пределах и чтобы отсутствовали сравнительно редкие явления, которые могли бы существенно исказить результаты.

Остальные проблемы включают измерение индексов и других интересующих нас переменных, выбор статистических тестов, которые должны быть применены к результатам. Заметим, что при измерении времени, затрачиваемого программистами и при регистрации отношения пользователей приходится брать информацию из личных журна-

Таблица 6.3

Результаты пяти сравнительных исследований пакетных и диалоговых режимов обработки

Работа	Индекс			
	Время программиста	Машинное время	Затраченное время	Предпочитаемый режим
Эриксон (1966)	1.9:1 (0.06)	3.4:1 (0.04)	—	Диалоговый
Голд (1967)	1.2:1 (0.07)	1:4.7 (0.001)	1:1	Диалоговый (0.001)
Грант и Сакман (1966)	1.6:1 (0.05)	1:1.4	—	Диалоговый
Шатцофф и др. (1967)	1:2.1 (0.02)	1.1:1	1.5:1 (0.08)	—
Смит (1967)	1.2:1	1:1.5	1.2:1	Пакетный с быстрым прохождением

Источник: Сакман (1968).
* Приведены отношения показателей пакетного режима к показателям диалогового. Статистическая значимость сравнения, если она известна, приведена в скобках. Например, в работе Эриксона показано, что отношение средних затрат времени программиста в пакетном режиме к средним затратам в диалоговом режиме составляет 1.9, а различие средних времен значимо на уровне 6%.

лов и анкет. Подробное обсуждение всех этих проблем и путей их решения можно найти в обзоре Сакмана (1968) и в статьях, описывающих каждое из пяти исследований.

Таблица 6.3 суммирует некоторые результаты этих пяти работ. Следует заметить, что исследование Смита на самом деле представляло собой сравнение обычного пакетного режима с пакетным режимом с быстрым прохождением заданий, а некоторые из обсуждавшихся выше проблем получили не вполне удовлетворительное решение в некоторых работах. Ни один индекс качества программ не приведен в этой таблице. В тех двух работах, где они измерялись, не обнаружено никакой значимой разницы между временами выполнения программ и между размерами программ. Голд (1967), однако, сообщил, что пользователи, работающие в диалоговом режиме, достигли существенно лучших решений и лучшего понимания своих проблем, чем работающие в пакетном режиме.

Из табл. 6.3 нельзя получить никаких определенных выводов, кроме выводов о предпочтении пользователей. Время, затрачиваемое программистом, проявляет тенденцию к уменьшению, а машинное — к увеличению при работе в диалоговом режиме. Результаты измерения расхода времени, отражающие объем работы, которую можно выполнить на установке, говорят скорее в пользу диалогового режима. Было най-

дено, что различия между отдельными пользователями по времени, затрачиваемому программистами, и машинному времени много больше, чем можно было бы объяснить режимом обработки (по трем работам, исследовавшим этот вопрос). Приводятся отношения от 3 : 1 до 14 : 1. По-видимому, квалификация программистов сильнее влияет на программный проект, чем режим обработки.

6.3.3. Сравнение поставщиков

Сравнение систем, предлагаемых различными поставщиками, является одной из самых сложных проблем в оценке производительности. Из-за различия технологии, архитектуры, стратегий управления и программ обслуживания пользователей сравнивать между собой существующие системы — это, как говорится, сравнивать «божий дар с яичницей». То, что системы трудно сравнивать друг с другом, особенно ясно видно, когда это сравнение предпринимается в терминах организационных параметров или индексов производительности оборудования, например длин слов, времен доступа, скоростей передачи данных, наборов команд и стратегий планирования. Проблема становится неразрешимой, если сравнение направлено на определение «наилучшей» системы среди нескольких независимо от применения, т. е. системы с наибольшей производительностью для любой возможной установки. Тем не менее некоторые индексы, ориентированные на оборудование и не зависящие от рабочей нагрузки, предлагались и использовались для сравнения систем. Перед обсуждением сравнений, основанных на выделенных в этой книге типах индексов, мы дадим краткий обзор таких подходов.

В области вычислительных систем имеется важное понятие, естественное, но трудно поддающееся определению и количественному измерению, это понятие *вычислительной мощности*. Если бы у нас было хорошее определение мощности, мы могли бы воспользоваться им для сравнения различных систем и для выбора наиболее удобной системы для установки с заданными требованиями к мощности. К сожалению, мы не располагаем таким определением. Наилучшим приближением к нему является емкость или максимальная пропускная способность, которая, однако, требует определения типов заданий и некоторых других дополнительных деталей. Было предпринято несколько попыток определить мощность, основываясь на характеристиках одного только оборудования, чтобы избежать углубления в детали операционных систем и рабочих нагрузок.

Самые ранние сравнения основывались на таких элементарных индексах, как, например, скорость внутренних часов ЦП, время сложения с фиксированной точкой, длительность цикла основной памяти, время выборки из памяти двух чисел, их сложения и записи результата. Примерами более сложных определений мощности являются оп-

ределения, введенные Шнейдевиндом (1966):

$$P = \frac{M}{t_{\text{цикл}}}, \tag{6.7}$$

где M — размер основной памяти в словах или символах, а $t_{\text{цикл}}$ — ее время цикла, и Грюнбергером (1966):

$$P = M\left(\frac{1}{t_{\text{сл}}} + \frac{1}{t_{\text{умн}}}\right), \tag{6.8}$$

где M измеряется в словах, а $t_{\text{сл}}$ и $t_{\text{умн}}$ — соответственно времена сложения и умножения. Заметим, что эти индексы все еще ориентированы на оборудование и независимы от рабочей нагрузки, как и более элементарные, упомянутые выше. Кроме того, они относятся исключительно к производительности комплекса «ЦП — основная память», который в течение продолжительного времени был наиболее дорогой частью системы, и поэтому вокруг него концентрировались все работы по оценке.

Рабочая нагрузка впервые появляется в индексах, основанных на смеси команд. Пример такого индекса — среднее время ЦП, затрачиваемое на выполнение одной команды:

$$P = \bar{v}_{\text{ЦП}} = \frac{1}{\bar{t}_x}, \tag{6.9}$$

где $\bar{t}_x$ — среднее время выполнения команды, задаваемое (5.9).

Другое определение мощности, требующее спецификации смеси команд, было использовано Найтом (1963):

$$P = \frac{[(L-7)\,Mk_1]^{k_2}}{k_3(\bar{t}_x + \bar{t}_{ion})}, \tag{6.10}$$

где L — длина команды в битах, M и $\bar{t}_x$ определены выше, $\bar{t}_{ion}$ — среднее время ввода-вывода на выполняемую команду (без перекрытия), k_1 — коэффициент, значение которого зависит от длины слова (фиксированная или переменная), k_2 имеет (в работе Найта) различные значения для научных и коммерческих установок, а k_3 — константа. Две различные смеси, соответствующие этим двум типам применений, были использованы Найтом для вычисления $\bar{t}_x$. Это различие также сохраняется в константах, входящих в довольно сложную формулу для оценки $\bar{t}_{ion}$. Эта формула выражает среднее время ввода-вывода, не перекрывающееся с работой ЦП, как функцию скоростей ввода-вывода и передачи данных из внешней памяти, времен пуска, остановки и возврата носителя информации и оценок коэффициентов использования устройств ввода-вывода. Отметим, что определение (6.10) выведено Найтом на основании оценок экспертов и использовалось в различных обширных исследованиях технического прогресса в области вычислительных систем (см. также Найт (1968) и Червены и Найт (1973)). Эти работы, в которых были исследованы сотни систем, введенных в строй в период между 1944 и 1972 годами, позволяют количественно описать *относительное удешевление* вычислительной техники.

Мы постулируем, что зависимость мощности P вычислительной системы (определение которой пока еще не приведено) от стоимости системы C имеет вид

$$P = KC^g, \tag{6.11}$$

где K и g — положительные константы. Таким образом, мощность на единицу стоимости равна

$$\frac{P}{C} = KC^{g-1}. \tag{6.12}$$

Мы говорим, что вычислительные системы представляют относительное удешевление, если их мощность возрастает быстрее, чем линейно с ростом стоимости. Следовательно, главным элементом нашего постулированного соотношения является значение g (см. рис. 6.1):

a) если $g>1$, имеем *относительное удешевление*;
b) если $g=1$, имеем *сохранение относительной стоимости*;
c) если $g<1$, имеем *относительное удорожание*.

Форма соотношения (6.11) была предложена в конце 40-х годов Грошем, который также предположил, что $g=2$. По этой причине соотношение (6.11) с $g=2$ называется *законом Гроша*. Одним из главных результатов, полученных Найтом, было экспериментальное доказательство закона Гроша, по крайней мере на протяжении рассматриваемого отрезка времени. Очень важно отметить, что в этом доказательстве было использовано данное в (6.10) определение P, которое является по существу индексом, основанным на характеристиках оборудования и смеси команд. В табл. 6.4 приведены результаты Найта (полученные линейным регрессионным анализом по методу наименьших квадратов) приблизительно о 500 системах, введенных в строй в период 1950—1972 гг. Отметим, что большее относительное удешевление получено для мини-компьютеров в более поздний период для научных прикладных систем.

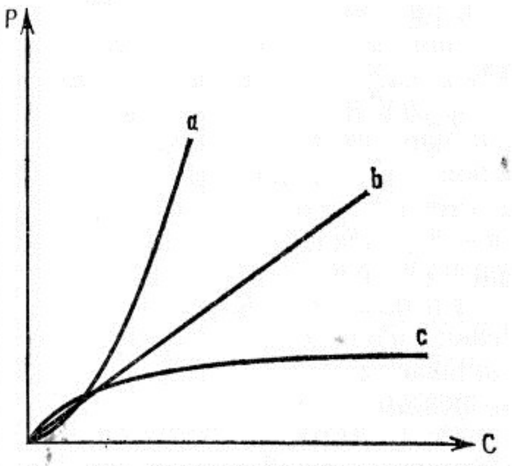


Рис. 6.1. Соотношение между вычислительной мощностью P и стоимостью C: a — относительное удешевление ($g>1$); b — сохранение относительной стоимости ($g=1$); c — относительное удорожание ($g<1$).

Ясно, что вопрос об относительном удешевлении очень важен для проектирования систем обработки информации для больших организаций, особенно если будущие пользователи географически разброса-

ны. Территория, на которой размещены пользователи, не обязательно должна иметь величину континента или страны, чтобы это утверждение было справедливо. Часто, даже если все они находятся в большом здании, проблема децентрализации заслуживает изучения. Основной вопрос, который возникает при этом, заключается в следующем: при

Таблица 6.4

Значения g в уравнении (6.11)

Период	Научные системы	Коммерческие системы
1950—1962	1.92	2.17
1962—1966	3.10	2.47
1964—1972 (мини-компьютеры)	5.88	—

Источники: Найт (1963), Найт (1968), Червены и Найт (1973).

заданных потребностях переработки информации в организации окажется ли большая централизованная вычислительная установка с рядом терминалов и удаленными станциями ввода удобнее, чем децентрализованная конфигурация, состоящая из многих систем, возможно размещенных в различных помещениях и соединенных в сеть? Очевидно, что этот вопрос, если возможно, должен решаться до выбора оборудования и проектирования системы или систем. Читатель, конечно, заметит, что эта дилемма сходна с возникающей при выборе однопроцессорной или многопроцессорной системы. Многопроцессорная система фактически имеет распределенную архитектуру подобно децентрализованной системе, но ее центральные процессоры обычно расположены вблизи друг друга, имеют доступ к одним и тем же блокам основной памяти и внешним устройствам и связаны более тесно. Таким образом, некоторые из аргументов в пользу или против децентрализации сохраняются в случае мультипроцессорной архитектуры, а некоторые нет.

Результаты, приведенные выше, показывают, что централизованная система обеспечивает заданную мощность за меньшую цену или большую мощность за ту же цену. Однако определение мощности (6.11), использованное Найтом, нельзя считать исчерпывающим, реалистическим описателем ценности вычислительной системы. Существует много дополнительных аргументов в пользу и против децентрализации, которые необходимо принять во внимание. При нашем обсуждении мы будем следовать подходу, предложенному Стритером (1973).

В табл. 6.5 подытожены наиболее важные потенциальные преимущества этих двух решений в случае системы обработки информации общего назначения. Большей части этих аргументов очень трудно или невозможно придать количественную форму. Мы теперь проиллюстри-

руем на примере простой приблизительный подход, описанный Стритером, учитывающий стоимость оборудования, линий связи, связи пользователя с машиной, прерываний обслуживания, а в качестве индекса производительности — среднее время прохождения задания.

Таблица 6.5

Потенциальные преимущества централизованного и децентрализованного подходов для вычислительной системы общего назначения

Преимущества централизации	Преимущества децентрализации
Относительное удешевление оборудования	Меньшая вероятность перерывов на обслуживание
Удешевление подготовки и сопровождения программ	Лучшее знание местных условий
Уменьшение степени дублирования базы данных	Более простое согласование программ и обслуживания с локальными требованиями
Уменьшение затрат на уход за оборудованием	Бóльшая гибкость в приспособлении к новым условиям
Уменьшение расходов на подготовку рабочего места	Большая заинтересованность персонала и лучшее стимулирование
Уменьшение затрат на охрану	Более тесный контакт персонала с местными пользователями
Более высокая степень использования оборудования из-за большого и более разнообразного коллектива пользователей	Легче и надежнее связь пользователей с системой
Более эффективное использование высококвалифицированного персонала	
Уменьшение среднего значения и дисперсии времен прохождения	
Большее разнообразие программ и средств обслуживания, доступных пользователям	
Объединение административных и технических служб	

Источник: Стритер (1973).

Пример 6.1. Вопрос о централизации рассматривается для системы обработки информации общего назначения, которую предстоит установить в большой фирме. Анализ потребностей фирмы дал приблизительную величину общей необходимой мощности P. Задача состоит в том, чтобы оценить удобство централизованного и децентрализованного решений как в терминах стоимости, так и в терминах производительности. Оба эти варианта представлены на рис. 6.2. Предположим, что в децентрализованном случае все системы идентичны и связаны полной сетью. Иначе говоря, каждая система соединена со всеми остальными. При такой крайне упрощенной формулировке задачи нашей конечной целью является выбор N, числа систем, которые будут установлены. Таким образом, мы попытаемся определить влияние N на общую стоимость C и на среднее время прохождения $\bar{t}$.

Предполагая справедливость закона Гроша, получим из (6.11) следующее уравнение для стоимости оборудования.

$$C_e = N\sqrt{\frac{P}{KN}} = K_e\sqrt{N}. \tag{6.13}$$

Для вычисления стоимости C_c линий связи, умножим их число $N(N-1)/2$ на стоимость одной линии, зависящую от потока сообщений, приходящегося на одну линию. Поток сообщений, приходящийся на одну линию, можно получить делением общего потока сообщений (пропорционального $N-1$, так как каждая модификация

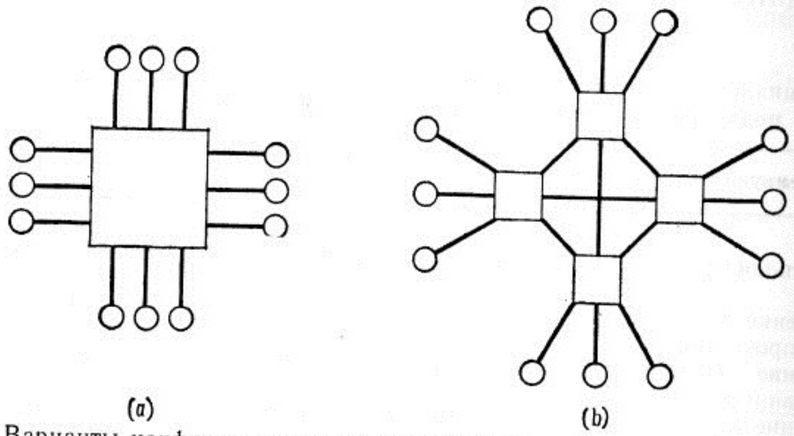


Рис. 6.2. Варианты конфигурации системы обработки информации; (a) — централизованная система ($N=1$); (b)—децентрализованная система с полными связями ($N=4$). Квадратами обозначены системы, окружностями — периферийные устройства ввода-вывода.

файла должна быть передана остальным $N-1$ системам) на число линий. Таким образом, поток сообщений на одну линию обратно пропорционален N. Если мы предположим квадратичное соотношение, как в законе Гроша, между потоком сообщений и стоимостью одной линии связи, квадрат стоимости одной линии связи будет обратно пропорционален N. Таким образом,

$$C_c = \frac{N(N-1)}{2} \cdot \frac{H}{\sqrt{N}} = K_c (N-1) \sqrt{N}. \tag{6.14}$$

Заметим, что для $N=1$ мы имеем $C_c=0$, как и следовало ожидать.

Стоимость связи пользователя с машиной можно считать пропорциональной (приблизительно) среднему расстоянию от машины до пользователя, которое, если пользователи и машины размещены географически равномерно, обратно пропорционально $\sqrt{N}$. Следовательно,

$$C_u = \frac{K_u}{\sqrt{N}}. \tag{6.15}$$

Наконец, стоимость прерываний обслуживания можно вычислить в первом приближении, предполагая, что отказы системы статистически независимы и что обслуживание прерывается только тогда, когда ни одна система не находится в состоянии работы. Если p_f — вероятность отказа системы и она предполагается одинаковой для всех систем, стоимость прерываний обслуживания C_i равна

$$C_i = K_i p_f^N. \tag{6.16}$$

Общая стоимость C, следовательно, определяется как сумма этих четырех стоимостей C_e, C_c, C_u, C_i:

$$C = K_e \sqrt{N} + K_c (N-1) \sqrt{N} + \frac{K_u}{\sqrt{N}} + K_i p_f^N. \tag{6.17}$$

Предположим, что в нашем проекте $P=10^6$ команд/с. Принимая $K=10^8$ команд$\times$долл.$^{-1}$, получим $K_e=0.1$ долл./с. Пусть, кроме того, $K_c=0.01$ долл./с, $K_u=0.1$ долл./с, $K_i=2.5$ долл./с и $p_f=0.1$. Уравнения (6.13—6.17) изображены на рис. 6.3 для этих численных значений. Очевидно, что общая стоимость минимальна

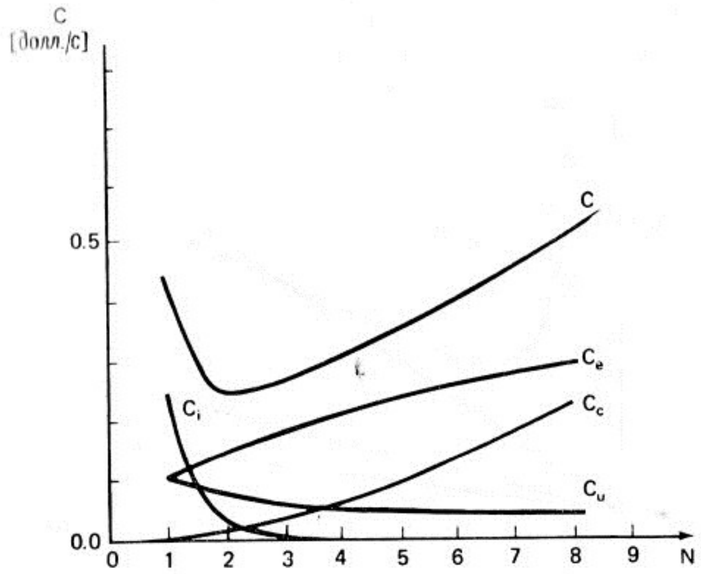


Рис. 6.3. Стоимости систем обработки информации в зависимости от числа установок.

для $N=2$, а стоимость для $N=3$ только немного выше, тогда как стоимость централизованной системы приблизительно в 1.8 раза выше минимума из-за относительно высокой стоимости прерываний обслуживания и низкой надежности систем.

Очень простой и грубый способ приблизительного определения влияния N на среднее время прохождения состоит в моделировании системы рис. 6.2 набором N различных систем с очередями и с одним обслуживающим прибором (рис. 6.4). Предположим, что все системы идентичны и одинаково нагружены, что поступление запросов является пуассоновским случайным процессом со средней интенсивностью λ/N, а времена обслуживания каждого прибора распределены экспоненциально со средним μ/N (так, что общая мощность обработки постоянна, поскольку сумма средних интенсивностей обслуживания всегда равна μ). В таком случае мы можем применить результаты, полученные в гл. 4 для $M/M/1$-моделей. В частности, уравнение (4.42) при средней интенсивности обслуживания μ/N и с коэффициентом использования $\rho=(\lambda/N)/(\mu/N)=\lambda/\mu$ дает

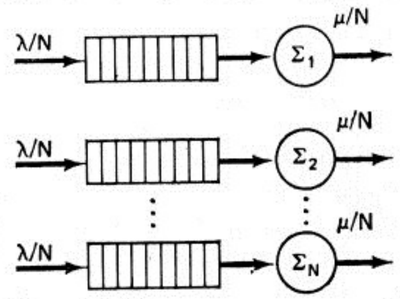


Рис. 6.4. Простая модель с очередями для системы рис. 6.2.

$$\bar{t} = \frac{N}{\mu(1-\rho)}. \tag{6.18}$$

Таким образом, среднее время прохождения $\bar{t}$ пропорционально N. На рис. 6.5 показаны некоторые графики $\bar{t}(N)$ для $\mu=1000$ заданий/ч при различных значениях λ, следовательно, ρ. Кривая $C(N)$, нарисованная на рис. 6.3, также перенесена на рис. 6.5, чтобы дать читателю наглядное представление компромисса между стоимостью и производительностью в этом крайне упрощенном исследовании. Согласно (6.18), за снижение общей стоимости на 44% при переходе от централизованной

системы к системе из двух установок приходится платить увеличением среднего времени прохождения на 100%.

Очевидно, что исследования проблем относительного удешевления и централизации, концептуально подобные рассмотренным выше, можно было бы проделать при помощи более всесторонних и более подхо-

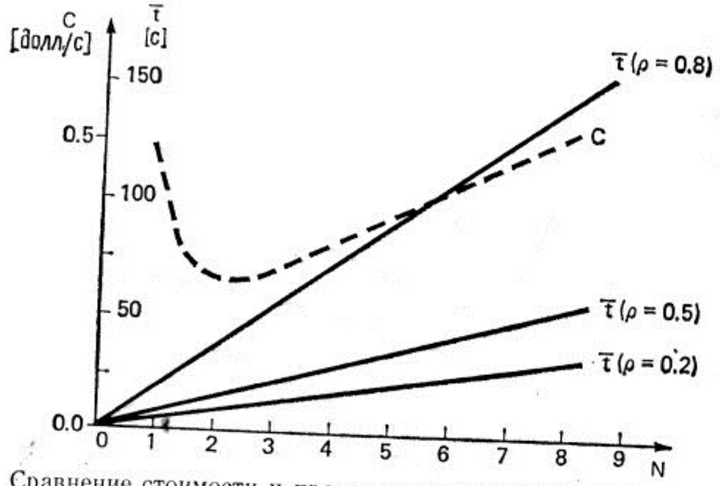


Рис. 6.5. Сравнение стоимости и производительности для систем на рис. 6.2.

дящих индексов производительности, чем введенные в этом разделе. Индексы, подобные определенным равенствами (6.7) и (6.8), имеют то преимущество, что зависят только от оборудования. Дополнительная зависимость от рабочей нагрузки, хотя и очень простая, поскольку включает только вычислительные аспекты рабочей нагрузки, характеризует индексы, подобные определенным в (6.9) и (6.10). Однако индексы производительности, которые мы ввели в разд. 1.4, по-видимому, гораздо ближе к тому, что действительно интересует администраторов и пользователей. Несколько упрощая, мы можем сказать, что администраторы наиболее заинтересованы в пропускной способности и коэффициентах использования, а пользователи системы — во времени прохождения или времени ответа. Эти индексы подвержены влиянию не только производительностей отдельных компонентов оборудования, но также и способов их соединения, координации их работы, взаимодействий пользователей с системой, трансляторов, программных средств, программ обслуживания и их производительностей. К сожалению, они также зависят от природы и размера рабочей нагрузки. Таким образом, эти индексы можно использовать в качестве индексов производительности системы, только если рабочая нагрузка, при которой получены их значения, задана ясно и однозначно. Целью работ по обоснованию предложений (шаг 3 процедуры выбора, описанной в разд. 6.2.1) является проверка того, что предлагаемая система обрабатывает данную рабочую нагрузку (или нагрузки) со значениями индексов производительности, удовлетворяющими обязательным тре-

ваниям. Такое требование может состоять в том, чтобы средняя пропускная способность была более 10 заданий в минуту, где «задания», названные в качестве единиц измерения пропускной способности,— специфические, точно определенные задания. В работах по сравнению производительностей рабочие нагрузки, для которых сравниваются значения индексов производительности, должны быть идентичными, чтобы эти значения можно было использовать в качестве критериев сравнения производительности систем (см. разд. 6.3.1).

Какую рабочую нагрузку следует выбрать для определения и оценки наших индексов производительности? Обычно, если установка уже существует, выбранная рабочая нагрузка совпадает с обычной максимальной производственной рабочей нагрузкой этой установки или близка к ней. Такой выбор позволяет также провести сравнение производительности предлагаемой системы с существующей. В более сложных исследованиях следует вносить поправку в эту рабочую нагрузку для учета воздействий, которые оказала бы каждая новая система на коллектив пользователей. Если установка не существует, начальная производственная рабочая нагрузка может быть оценена на основании имеющихся данных о потребностях пользователей в переработке информации. Часто рассматривается также одна или несколько проектируемых нагрузок как для существовавшей ранее, так и для новой установки. Задание этих существующих и будущих рабочих нагрузок крайне затруднено требованием их идентичности для всех сравниваемых систем. Это подразумевает портативную модель рабочей нагрузки, не зависящую от поставщика.

Но какими моделями рабочей нагрузки, вероятнее всего, придется пользоваться? Чтобы ответить на этот вопрос, мы должны определить, насколько удобен для задач выбора каждый метод оценки. Как мы увидим в разд. 6.4, в этом случае применимы методы измерения, так как различные варианты, среди которых придется делать выбор, предполагаются существующими (если не в точности такими, то по крайней мере в похожих конфигурациях или версиях) и доступными поставщику или пользователю. Кроме того, эти методы оказываются обычно наиболее подходящими. Выбранные рабочие нагрузки, как существующие, так и проектируемые, не могут использоваться непосредственно для управления сравниваемыми системами (а проектируемые к тому же еще и не существуют). Таким образом, должна использоваться исполняемая искусственная модель. Среди всех типов исполняемых искусственных моделей, обсуждавшихся в разд. 5.3.4 и 5.3.5, наиболее широко применяются в исследованиях по выбору те, которые состоят из наборов эталонов или синтетических заданий. Однако использовались также и более элементарные модели, например смеси команд и метод ядер.

Для того чтобы требования производительности, сформулированные в заявке, имели смысл, следует включать в заявку модель (модели) рабочей нагрузки или ее подробную спецификацию. Часто заказчик разрабатывает набор эталонов или синтетических заданий, а

каждый поставщик приспосабливает его к предлагаемой системе. Время выполнения этого набора программ на данной системе, т. е. мера ее средней пропускной способности, обычно служит в качестве основного теста производительности на шаге обоснования предложения (шаг 3). Это также наиболее распространенный индекс при сравнении производительностей, хотя проводятся и другие измерения во время преобразования и выполнения программ. Это иллюстрируется следующим примером, который отражает суть работы по сравнению производительностей, описанной Страусом (1972).

Пример 6.2. Нужно сравнить производительности четырех новых систем, предложенных для университетского вычислительного центра (как друг с другом, так и с уже установленной системой). Пять рассматриваемых систем и их конфигурации перечислены в табл. 6.6 (A — существующая система). В качестве первичного индекса производительности выбрана средняя пропускная способность каждой из этих

Таблица 6.6

Системы и конфигурации, сравниваемые в примере 6.2

Система	Изготовитель и модель	Размер памяти	Размер памяти на диске
A	IBM 360/50	256*K* байт, 2 мкс 1*M* байт, 2.5 мкс	484*M* байт (2314)
B	Burroughs B6714	128*K* слов, 48 бит/слово	484*M* байт
C	IBM 370/155	768*K* байт	600*M* байт (3330)
D	UNIVAC 1108	196*K* слов, 36 бит/слово	80*M* слов
E	XDS SIGMA 9	512*K* байт	600*M* байт (2314)

Система	Число лентопротяжных устройств	Операционная система	Условия поставки	Ориентировочная стоимость в месяц (в долл.)
A	6	OS/MFT	Аренда на 1—2 года	31 000
B	6	MCP	Аренда на 1 год	28 600
C	6	OS/MFT OS/MVT	Аренда на 1—2 года	39 900
D	6	EXEC 8	Продажа в рассрочку на 5 лет	33 000
E	4	UTS	Продажа в рассрочку на 5 лет	30 000

Источник: Страус (1972)

систем под управлением рабочей нагрузки установки. В качестве исполняемой модели этой рабочей нагрузки разработан набор эталонов. Характеристики набора приведены в табл. 6.7. В набор включено, кроме того, дополнительное задание, не упомянутое в таблице: *пустое задание*, которое никогда не кончается и которое использует все время ЦП, не израсходованное остальными заданиями или операционной системой, таким образом можно выявить дополнительные неиспользованные возможности

системы. Четыре задания на языке ФОРТРАН связаны с большим объемом вычислений и проверяют точность работы ЦП и различных стандартных математических функций и подпрограмм. Остальные восемь заданий на ФОРТРАНе предназначены для проверки эффективности программ ввода-вывода на языке ФОРТРАН (это одна из областей, в которых система A заведомо неэффективна). Одно из заданий на языке КОБОЛ создает индексно-последовательный тестовый файл для четырех

Таблица 6.7

Характеристики набора эталонов для примера 6.2

Язык	Число заданий	Времена ЦП IBM 360/50 (с)		
		Компиляция	Выполнение	Общее
ФОРТРАН	12	82	2229.8	2311.8
КОБОЛ	6	447.8	92.8	540.6
WATFIV	6	210.9	—	210.9
	24	740.7	2322.6	3063.3

Операционные правила

П1. Не допускается оптимизация исходного текста программы.
П2. Должны быть сделаны два основных прогона (прогон 1 и прогон 2); можно привести результаты других, оптимизированных прогонов.
П3. В прогонах 1 и 2
a) все задания остаются в заданных классах (если они пригодны),
b) максимальная степень мультипрограммирования должна равняться 5,
c) порядок заданий на входе фиксирован и совпадает с заданным,
d) всем заданиям назначены одинаковые внешние приоритеты.
П4. Прогон заканчивается, когда завершены все задания, кроме пустого.
П5. В прогоне 1 выполнение начинается после того, как считаны все задания; должны быть зафиксированы времена чтения, выполнения и печати.
П6. В прогоне 2 выполнение начинается как можно быстрее; должно быть зафиксировано общее время выполнения.

Выходные документы

Описание точной конфигурации оборудования и программного обеспечения
Все листинги печатающих устройств и протокол операторского терминала
Времена ЦП, моменты начала и конца обработки для каждого задания и каждого шага задания
Вся стандартная учетная информация
Все необходимые изменения первоначального текста программы
Все затраты человеческого и машинного времени до и во время прогона
Результаты оптимизированных прогонов

Источник: Страус (1972).

заданий, являющихся программами для обработки административной рабочей нагрузки. Шестая программа на КОБОЛе содержит ошибки и должна проверить диагностические возможности компилятора с КОБОЛа. Эталоны на языке WATFIV являются маленькими студенческими заданиями, которые, грубо говоря, считаются учебными, задания на ФОРТРАНе, поступающие от студентов и аспирантов, считаются исследовательскими, а задания на КОБОЛе — административными. Читатель уже определенно заметил, что при конструировании этой модели рабочей нагрузки бы-

ло уделено сравнительно мало внимания требованию представительности, по крайней мере в смысле использования физических или логических ресурсов. Действительно, большая часть эталонов выбиралась в целях, отличных от сравнения относительного использования ресурсов производственной рабочей нагрузкой. Мы уже видели в разд. 5.2.2, что требование представительности присутствует даже в том случае,

Таблица 6.8

Некоторые результаты сравнительного исследования, описанного в примере 6.2

Индекс	Система A	B[1]	C[1]	D[2]	E[2]
Общий зафиксированный расход времени (мин)	94.9	43.46	14.7	21.25	32.4
Общее израсходованное время минус время свободного процессора (мин)	61.6	43.4	14.3	17.7	18.1
Оценка общего расхода времени минус время свободного процессора без изменений эталонов (мин)	61.6	43.4	14.3	19.1	63.7
Общее время ЦП минус время свободного процессора (мин)	51.05	40.2	11.75	16.3	14.5
Время программиста на преобразование информации (ч)	40[3]	80	4	160	40
Машинное время на преобразование (ч)	6[3]	20[4]	20	5	3
Средняя пропускная способность на единицу стоимости ($10^6 \times$задания/(мин $\times$ $\times$долл./мес))	12.57	19.33	42.06	41.09	44.19

Источник: Страус (1972), с разрешения AFIPS, Inc.
[1]) Приоритеты были подобраны так, чтобы пустое задание могло выполняться только тогда, когда не может выполняться никакое другое задание.
[2]) В некоторых эталонах были сделаны изменения текста программы на входном языке; их ожидаемый эффект исключен из цифр, приведенных в третьем индексе этой таблицы.
[3]) Времена подготовки эталонов.
[4]) На модели 370/145 с учетом перевода на ANS COBOL, который не потребовался.

когда из-за частных целей исследователя параметры, характеризующие рабочую нагрузку, не ориентированы на ресурсы. Следует также вспомнить важное замечание, сделанное в разд. 6.3.1, что в сравнительных исследованиях степень представительности модели рабочей нагрузки может быть ниже, чем в работах по анализу производительности, при условии что различия значений индексов производительности относительно нечувствительны к отличиям модели рабочей нагрузки от реальной рабочей нагрузки.

Главная цель разработки операционных правил, приведенных в табл. 6.7, состояла в том, чтобы повысить степень справедливости сравнения, пытаясь выполнять эталоны в достаточно сходной операционной среде, не отдавая предпочтения какой-либо одной из них. Это крайне трудная проблема, одна из тех, с которыми исследователь вынужден сталкиваться непосредственно (ему приходится сравнивать «божий дар с яичницей»). Запрет оптимизировать программы в кодах (правило 1) является, с одной стороны, удобным, с другой — реалистичным. Однако результаты, сообщенные поставщиком системы E, показывают, что простая замена оператора WRITE на оператор CALL BUFFOUT в тексте эталонов на языке ФОРТРАН уменьшает общие затраты времени системы E приблизительно в 3.5 раза (см. табл. 6.8). Итак, если система E будет выбрана и установлена, реально ли предположить, что программисты быстро

научатся использовать CALL BUFFOUT вместо WRITE? С другой стороны, существуют очевидные оптимизации программ, которые нельзя разрешать, по крайней мере если мы подозреваем, что поставщики, вероятно, знают больше о своих системах, чем их типичные пользователи. Аналогичные соображения можно высказать относительно правил, которые должны уменьшить влияние различий в размере памяти (П1б) и предотвратить расположение эталонов поставщиками в порядке, наилучшим образом приспособленном к их стратегиям размещения (П3a, c, d). Заметим, что поставщики систем B и C не соблюдают правило о приписывании равных внешних приоритетов всем заданиям (правило П3d), чтобы обеспечить выполнение пустого задания только в том случае, когда нет другого задания, готового к выполнению. Правила П4, П5 и П6 предназначены для точного определения израсходованного времени, так чтобы сравнение его значений имело смысл. Следует заметить, что разрешение вспомогательных прогонов, содержащееся в правиле П2, позволяет поставщикам представить более полную картину предлагаемых ими систем посредством устранения в этих прогонах ограничений, которые, по их мнению, ставят их в слишком невыгодное положение.

Наборы выходных документов, предоставляемых каждым поставщиком и перечисленных в табл. 6.7, были продиктованы целями исследования. Например, исследователей интересовали полнота, достаточность и точность учетной информации, пре-

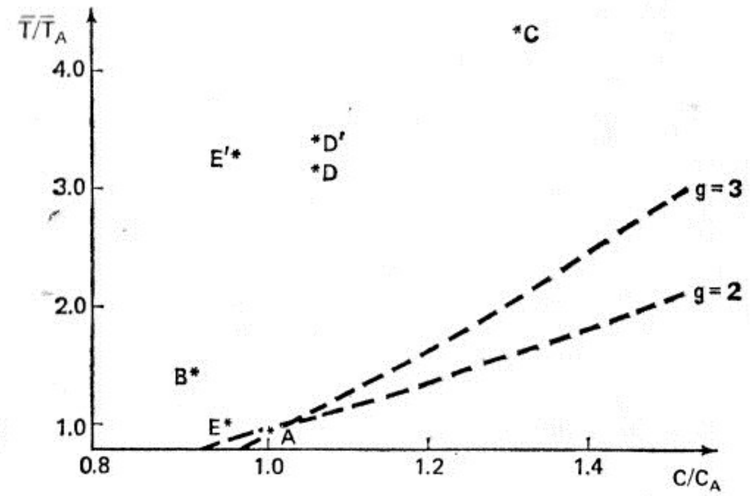


Рис. 6.6. Средняя относительная пропускная способность (во время выполнения 64 эталонов, описанных в примере 6.2) как функция приблизительных относительных ежемесячных затрат для 5 систем, приведенных в табл. 6.6. Две пунктирные линии — нормализованные графики уравнения (6.11') для $g=2$ (закон Гроша) и для $g=3$. Точки D′ и E′ соответствуют прогонам с модифицированными эталонами.

доставляемой пользователям. Они также интересовались стоимостью и временем, необходимым для перехода на новую систему, и пользовались цифрами, приведенными каждым из поставщиков для преобразования эталонов, как грубой относительной мерой сложности преобразования.

Некоторые из наиболее интересных результатов этого исследования приведены в табл. 6.8. Из-за сомнений, выраженных выше, относительно справедливости операционных правил и из-за того, что не все поставщики их соблюдали, интерпретация результатов не так однозначна, как хотелось бы. С точки зрения производительности система C оказалась победителем. Она, однако же, и самая дорогая. За системой C следует D, которая по стоимости занимает второе место и требует максимальных затрат человеческого труда на преобразование информации. Соотношения между стоимостью и производительностью этих пяти систем, основанное на затраченном времени (вторая и третья строка табл. 6.8) и на стоимости (последний столбец табл. 6.6), по-

казано на рис. 6.6. Пунктирные линии соответствуют уравнению (6.11), нормированному по отношению к цене и производительности системы A. То есть мы изобразили функцию

$$\frac{P}{P_A}=\left(\frac{C}{C_A}\right)^g, \tag{6.19}$$

где P совпадает с нашим индексом производительности $\bar{T}$, средней пропускной способностью для набора из 24 эталонов, описанных в данном примере. Очевидно, что если бы можно было ожидать такого же относительного удешевления, как определенное Найтом, для систем, сравниваемых в этом исследовании (и при том же определении производительности), то значительное отклонение экспериментальных точек от кривых на рис. 6.6 можно было бы объяснить только тем, что предлагаемые системы (кроме системы E при выполнении немодифицированных эталонов) обладают другим значением K, так что уравнение (6.19) следует в действительности написать в виде

$$\frac{P}{P_A}=\frac{K}{K_A}\left(\frac{C}{C_A}\right)^g. \tag{6.20}$$

Отношение K/K_A, равное, судя по диаграмме рис. 6.6, по меньшей мере 2, можно рассматривать как грубую меру технического прогресса в период между вводом в эксплуатацию системы A и созданием остальных систем.

Даже если бы мы были полностью убеждены в справедливости этих правил и в их детальном соблюдении всеми поставщиками, интерпретация результатов все еще оставалась бы трудной. Какую систему нам следует выбрать, после того как мы установили довольно общеизвестный и не удивительный факт, что системы с более высокой производительностью являются и более дорогими? Решение о выборе должно принимать во внимание множество факторов, а не только стоимость и производительность каждой из предложенных систем. Большинство этих факторов упоминалось в разд. 6.2.1. Если мы ограничим наше внимание только стоимостью и производительностью, было бы полезно рассматривать значения *индекса стоимость — производительность*, например средняя пропускная способность на доллар. Из последней строки табл. 6.8 видно, что система E′ (с модифицированными эталонами) является наилучшей в смысле стоимости и эффективности. За ней следуют системы C и D′ на расстоянии, которое легко может быть отнесено на счет различных ошибок, несомненно присутствующих в процессе оценки. ■

Следует отметить, что сравнение производительностей в примере 6.2 было проведено с использованием значений затраченного времени, измеренных поставщиками без проверки статистической значимости или оценки доверительных интервалов, как это описано в разд. 6.3.1. Это пренебрежение оправдано целями исследования и выбранным индексом производительности. Средняя пропускная способность (являющаяся средним по времени, а не средним по множеству заданий) на самом деле вычисляется по единственному измерению общего затраченного времени, а на единичном образце можно выполнить мало статистических проверок. Изучение воздействия на этот индекс со стороны не полностью контролируемых источников изменения было бы полезно, но и крайне дорого, так как оно потребовало бы набора тестов, которые нужно прогонять большое количество раз. Более того, тот факт, что эксперименты выполняются различными поставщиками с использованием разных измерителей и методов, по-видимому, внесет в результаты систематические ошибки, большие, чем те, что можно обнаружить и измерить с помощью дисперсионного анализа.

Пример 6.2 определенно вызовет у читателя вопросы, относящиеся к определению и построению тестов, пригодных для переноса с системы на систему. Эти проблемы обсуждаются в разд. 6.5.

6.3.4. Сравнение установок

Когда пользователь или организация имеет дело с таким применением компьютера, для которого неэкономично или неразумно создавать отдельную установку и нет готовой системы, пригодной для этого применения, возникает проблема выбора наиболее подходящей установки среди доступных или подходящего бюро обслуживания. Эта проблема существенно отличается от проблемы выбора поставщика, рассмотренной в разд. 6.3.3. При этом мы приобретаем не элементы аппаратного или программного обеспечения, а только услуги. Точка зрения принимающих решение непосредственно отражает точку зрения пользователя (что касается производительности, принимаются во внимание только аспекты реактивности, а не продуктивности и не коэффициента использования). Единственными входными данными для системы, которые известны пользователям, являются их программы, составляющие только часть рабочей нагрузки каждой системы. Подобные ситуации возникают при различных типах операционной среды. Например, программист или бригада, принадлежащие к организации с многими установками общего назначения или имеющей доступ к вычислительной сети, должны решать, какую установку использовать для данного применения. Другой пример — компания, которой предстоит выбрать, к какой службе разделения времени или вообще к какому поставщику машинного времени присоединиться. Иногда предстоит выбор между бюро обслуживания и приобретением собственной вычислительной системы подходящего размера и стоимости. Эта интересная проблема может включать сравнение способов обработки и использование некоторых из подходов, описанных в предыдущем и данном разделах. Мы оставим ее читателю в качестве предмета для дальнейших размышлений (см. задачу 6.12). Случай, когда в момент принятия решения неизвестны программы, которые предстоит выполнять на выбранной установке, не будет обсуждаться. Мы ограничимся простой ситуацией, когда все программы существуют и написаны на языке высокого уровня. Разумеется, стоимости их прогона и выгоды, ожидаемые от этой работы, должны быть настолько велики, чтобы оправдать исследование.

Каковы аспекты вычислительной системы, наиболее интересующие пользователя при рассмотрении проблемы выбора установки? Морган и Кэмпбелл (1973) перечисляют следующие характеристики: легкость использования (легкость доступа, наличие служб консультации и документации), доступность (вероятность, что система в данный момент времени находится в рабочем состоянии), надежность (вероятность

возникновения ошибки и восстановления системы в течение определен-
ного времени), гибкость (диапазон применений и языки), производи-
тельность (время прохождения или ответа), стоимость, технологиче-
ский риск (риск, что система новая и, следовательно, недостаточно
проверенная) и риск планирования (вероятность, что система будет
снята с производства прежде, чем будут удовлетворены потребности
пользователя по обработке). Для большинства этих аспектов лучшим
источником информации для пользователя системы является реальное
использование. В частности, характеристики стоимости и производи-
тельности можно оценить гораздо более надежно при прямых экспе-
риментах на каждой из систем-кандидатов, чем любым другим мето-
дом. Как предположили Морган и Кэмпбелл, наиболее эффективный
способ измерения стоимостей и производительностей различных уста-
новок по отношению к данному набору программ состоит, по-види-
мому, в том, чтобы создать исполняемую модель этого набора программ,
закодировать их для каждой системы и выполнить. Так как мы пред-
полагаем, что программы существуют, их можно использовать в этом
эксперименте непосредственно. Однако цены переноса их и их баз дан-
ных на каждую систему, а также стоимости выполнения на каждой
из систем оказались бы во многих случаях недопустимыми. Ясно, что
в этих случаях очень важна компактность выполняемой модели. Модели
программ в виде «синтетических заданий», вообще говоря, можно пре-
образовать дешевле, чем модели из эталонов (см. разд. 5.3.5), поскольку
ку не нужно преобразовывать реальные файлы, используемые реаль-
ными программами. Синтетические задания могут создавать файлы,
содержимое которых не имеет никакого значения, и получать к ним
доступ. Кроме того, синтетические задания гораздо более гибки, чем
эталоны, и если они настроены должным образом, то могут быть бо-
лее представительной моделью ограниченного, точно определенного
набора программ, чем комбинации реальных программ, выбранных из
этого набора.

Как нам построить синтетическую модель наших программ? Мо-
дель должна быть системно-независимой и представительной. Систем-
ная независимость в этом контексте означает, что модель должна
быть выполнима и равно представительна на всех сравниваемых си-
стемах. Это в точности совпадает с требованием, которому должны
удовлетворять модели рабочей нагрузки для сравнения поставщиков.
Представительность можно определить на двух различных уровнях.
Первый уровень сравнительный. Модель будет представительна, если
порядок, в котором можно расположить системы по производитель-
ности и стоимости на основе результатов ее использования, совпадает
с порядком, полученным на самих моделируемых программах. Жела-
тельно в целях анализа «стоимость — производительность» иметь мо-
дель, сохраняющую не только упорядоченность, но и приблизительные
отношения стоимостей.

Второй уровень представительности относится к возможности
оценивать действительную производительность и действительную стои-

мость по результатам моделирования. Эта дополнительная степень
обоснования модели, конечно, желательна, но не всегда существенна.
Морган и Кэмпбелл (1973) показали, что второй уровень представи-
тельности достижим при ограничениях, обусловленных нестабиль-
ностью большинства алгоритмов загрузки ресурсов. Что касается
производительности, то результаты, опубликованные Беллом (1974),
подтверждают интуитивно понятный факт, что экспериментальные срав-
нения в этой области ограничены и оказываются либо очень грубыми,
либо очень дорогими. Самая большая проблема состоит в том, что ин-
тересующие нас индексы производительности, например время про-
хождения задания, очень чувствительны к рабочей нагрузке, сущест-
вующей в системе в момент их измерения, а экспериментатор в этом
случае вообще не может ни наблюдать рабочую нагрузку, ни управ-
лять ею. Белл (1974) обнаружил некоторое непостоянство израсхо-
дованного времени и времени ЦП синтетического задания, даже если
задание выполнялось одно в системе. Разумеется, это непостоянство
значительно возрастает, если задание выполняется в мультипрограмм-
ном режиме вместе с другими заданиями. Имело бы смысл в предпо-
ложении стационарного поведения взять в качестве индекса произ-
водительности распределение времен прохождения (или его описате-
ли, например среднее и дисперсию). Однако оценивать эти описатели
или хотя бы только среднее по выборочным распределениям в общем
случае очень дорого. Подходы, которые могли бы уменьшить изменчи-
вость, обусловленную остальной частью рабочей нагрузки, состоят
в том, чтобы выполнять синтетическое задание с наивысшим возмож-
ным приоритетом или предлагать его в момент, когда нагрузка мала.
Однако эти эксперименты не могут предоставить пользователю соот-
ветствующую информацию о производительности, состоящую из диа-
пазона времен прохождения, которые следует ожидать при «нормаль-
ных» условиях. Таким образом, от идеи точного сравнения производи-
тельностей в большинстве случаев отказываются и заменяют ее менее
формальными планами. Пользователь может получить информацию
о производительности, проводя эксперименты на каждой системе, оп-
рашивая других пользователей и изучая имеющиеся статистические
данные по установкам.

Чтобы проиллюстрировать, как можно выполнить сравнение стои-
мостей, представим в следующем примере подход, сходный с исполь-
зованным в работе Моргана и Кэмпбелла.

Пример 6.3. Необходимо сравнить две установки A и B с точки зрения прикладной программы, которая должна быть выполнена многократно. Требуется оценить и сравнить стоимость каждого выполнения программы. Прежде всего рассмотрим формулы стоимости прогона задания для этих двух установок. Для простоты предположим, что обе формулы имеют структуру

$$C_X = \sum_{i=1}^{n_X} c_{Xi} u_{Xi} \quad (X = A,\ B), \tag{6.21}$$

где n_X — число ресурсов, запрошенных пользователями установки X, c_{Xi} — стоимость единицы ресурса i на X, а u_{Xi} — число единиц ресурса i на X, необходимое для этого задания. Пусть на установке А назначены цены за использование ЦП (единица : 1 с процессорного времени), основной памяти (единица : одно слово основной памяти, запрошенное заданием), диска (единица : одно обращение к диску), лентопротяжных устройств (единица : установка одной бобины на лентопротяжное устройство), устройства чтения перфокарт (единица : одна прочитанная перфокарта) и печатающего устройства (единица : одна отпечатанная строка). Предположим также, что в установке В назначены цены только за использование ЦП, основной памяти, диска и лентопротяжного устройства (с другой единицей : одно обращение к ленте). Следовательно, модель нашей прикладной программы окажется уменьшенным представлением программы с точки зрения потребления упомянутых выше ресурсов на каждой установке. Предположим, что одно выполнение этой программы займет 350 с времени ЦП на системе А и 280 с на системе В; тогда, если масштабный коэффициент r равен 100, модель должна израсходовать соответственно 3.5 с и 2.8 с времени ЦП на установках А и В. Такое же соотношение должно существовать для требуемого размера памяти, чисел обращений к дискам и лентам, количества установленных лент, числа прочитанных перфокарт и отпечатанных строк. Вообще, нам бы подошла модель, использующая u_{Xi}/r единиц ресурса i системы X ($i=1, 2, \ldots, n_X$), так чтобы ее стоимость оказалась C_X/r. Если это так, то оценку для C_X можно получить умножением этой стоимости на r. Заметим, что не обязательно, чтобы модель была одинакова для всех установок. В принципе мы могли бы для каждой системы сконструировать свою модель со своим масштабным коэффициентом. Из-за различия формул стоимости даже в простейших рассматриваемых нами случаях может потребоваться введение некоторых различий между моделями. Например, если формула для системы В содержала бы член $c_{B1}u_{B1}u_{B2}$ (где u_{B1}, например, израсходованное системой время ЦП, а u_{B2} — запрошенный размер памяти), то потребовалось бы, скажем, приравнять u_{B2} в модели величине u_{B2} в программе, а не этой же величине, деленной на r; иначе вклад этого члена в стоимость выполнения модели на системе В оказался бы в r раз меньше, чем следует. Заметим также, что стоимость задания предполагалась зависящей только от общего потребления ресурсов заданием. Это не всегда так, но часто это предположение можно считать разумным, особенно если ресурсы, входящие в формулу стоимости, таковы, что их использование заданием не зависит от поведения остальной части рабочей нагрузки. В любом случае это предположение означает, что порядок поступления запросов на ресурсы в модели несущественен. Единственно важное требование при конструировании модели состоит в том, чтобы было известно соотношение между общим количеством ресурсов, запрошенных моделируемой программой, и той же величиной для модели.

Требование системной независимости подразумевает, что программу следует характеризовать параметрами, относящимися скорее к логическим, нежели к физическим ресурсам (см. разд. 5.2.3). Другими словами, модель должна представлять одну и ту же программу (более точно, одну и ту же задачу обработки информации) для обеих систем А и В. Таким образом, она не может содержать параметров, относящихся к ресурсам А, которыми В не обладает (например, число слов медленной памяти, необходимых программе, если А оборудована такой памятью, а В нет). Кроме того, параметры модели не должны зависеть от способов реализации ресурсов в обеих системах. Например, если у этих двух систем различны аппаратно-программные механизмы доступа к файлам, этот аспект деятельности задания следует характеризовать числом (и, возможно, историей) обращений к операционной системе, а не числом (и историей) обращений к диску (которые будут зависеть от системы).

Подход, предложенный Морганом и Кэмпбеллом (1973) направлен на удовлетворение трех требований, а именно компактности, представительности и независимости от системы, при конструировании уменьшенной копии программы в терминах *запросов на ресурсы на языке высокого уровня*. Компилятор рассматривается как преобразователь запросов на ресурсы на языке высокого уровня в соответствующие запросы физических ресурсов каждой машины или точнее ресурсов системы. Эти запросы в свою очередь транслируются в запросы физических ресурсов операционной системой. К ресурсам языка высокого уровня относятся те ресурсы (Морган и Кэмпбелл

называют их *услугами*), которые известны программисту и запрашиваются им. Если такой является, скажем, ФОРТРАН, мы могли бы их назвать ресурсами машины ФОРТРАНа. Таблица 6.9 содержит список этих ресурсов и их главные характеристики, которые можно использовать для определения подходящих подтипов ресурсов. Если бы наша прикладная программа была программой на ФОРТРАНе из табл. 6.10, мы могли бы проанализировать ее, как показано в табл. 6.11 (которая может быть на-

Таблица 6.9

Типы ресурсов ФОРТРАНа

Тип ресурса	Основные характеристики
1. Арифметические и логические операции	Тип оператора (присваивание, цикл, ветвление) Тип данных (действительные, целые, комплексные) Операции (=, +, —, *,/, .AND., .OR.)
2. Вызовы подпрограмм и функций	Передаваемые параметры
3. Индексация массивов	Число индексов
4. Вызов подпрограмм при выполнении	Явный (SQRT, FLOAT, INT), неявный (преобразования данных)
5. Операции ввода-вывода	Тип устройства Характеристики файла
6. Область памяти	Применение (глобальное, локальное) Тип данных (целые, действительные, двойной точности, комплексные)

Источник: Морган и Кэмпбелл (1973).

Таблица 6.10

Программа на ФОРТРАНе для моделирования

Текст	Тип ресурса (см. табл. 6.9)	Кратность выполнения
DIMENSION A(1000)	6	—
TOT = 0.	1	1
DO 10 I = 1,1000	1	1000
READ (5,500) A(I)	3,5	1000
500 FORMAT (F6.2)	4	1000
10 TOT = TOT + SQRT(A(I))/A(I)	1,3,4	1000
AVG = A(1)/2.0	1,3	1
DO 20 I = 2,1999	1	1998
20 AVG = AVG + (A(I/2+1) + A((I+1)/2))/2.0	1,3	1998
WRITE (6,600) TOT, AVG	5	1
600 FORMAT (7HANSWER = ,2F10.2)	4	1
STOP	—	1
END	—	—

Источник: Морган и Кэмпбелл (1973).

звана *смесью ресурсов для ФОРТРАНа*), и сконструировать ее синтетическую модель, показанную в табл. 6.12. Эта модель воспроизводит запросы ресурсов ФОРТРАНа программой из табл. 6.10, уменьшенные в 10 раз. После трансляции компиляторами с ФОРТРАНа систем A и B эта модель будет выполнена на обеих системах, а сов

Таблица 6.11

Ресурсы ФОРТРАНа, затребованные программой из табл. 6.10

Тип ресурса (см. табл. 6.9)	Подтип ресурса	Кратность затребования программой	Кратность затребования моделью (масштаб:10)	Относительная частота (%)
1	a) Действит. присваивание	1+1000+1+1998=3000	300	9.68
	b) Цикл	1000+1998=2998	300	9.67
	c) Действит. сложение	1000+1998+1998=4996	500	16.12
	d) Целое сложение	1998+1998=3996	400	12.89
	e) Действит. деление	1000+1+1998=2999	300	9.67
	f) Целое деление	1998+1998=3996	400	12.89
2	—	—	—	0.00
3	Одномерная индексная	1000+1000+1+1998+ +1998=5997	600	19.35
4	a) Явный вызов подпрограммы (SQRT)	1000	100	3.22
	b) Неявный вызов подпрограммы преобразования символов в действ. числа	1000	100	3.22
	c) Неявный вызов подпрограммы преобразования действ. чисел в символы	2	0	0.00
5	a) Чтение (устройство чтения перфокарт)	1000	100	3.22
	b) Печать (печатающее устройство)	1	0	0.00
6	a) Локальные действит. переменные	1002	100	—
	b) Локальные целые переменные	1	0	—

Источник: Морган и Кэмпбелл (1973).

мость выполнения программы из табл. 6.10 можно оценить, умножив цены в модели на 10. Результаты эксперимента, представленные Морганом и Кэмпбеллом для двух программ и четырех систем, показывают, что эти оценки стоимости были ниже реальных не более чем на 7%.

С этим подходом связано несколько проблем. Одна из них заключается в том, что при сложных формулах стоимости конструирование представительной модели может оказаться затруднительным (особенно по отношению к оценке использования основной памяти, которая могла бы потребовать воспроизведения истории размещения программы в памяти). Другая проблема связана с объемом работы, необходимой для извлечений соответствующей информации из больших прикладных программ. В большинстве случаев необходимо измерение *профиля программы*, т. е. получение информации о числе выполнений каждого оператора в типичном прогоне, поскольку

ресурсы ФОРТРАНа, затребованные каждым оператором программы, должны быть [illegible] на ожидаемую частоту выполнения этого оператора. Анализ каждого оператора ФОРТРАНа для определения затребованных им ресурсов оказывается громоздким, даже несмотря на то что информация, заключенная в профиле, позволяет нам

Таблица 6.12

Синтетическая модель программы из табл. 6.10 для сравнения установок

Операторы	Тип и подтип ресурса (см. табл. 6.11)	Число выполнений
DIMENSION A(100)	6a	—
A(1)=10	1a, 3	1
DO 10 I=1,100	1b	100
READ (5,500) X	5a	100
500 FORMAT (F6.2)	4b	100
Y=(SQRT(2.0)+75.0)/63.2	1a, 1c, 1e, 4a	100
DO 20 J=1,2	1b	200
20 A(J/2+2)=(Y+A(J))/(Y — A((J+1)/2))	1a, 1c, 1d, 1e, 1f, 3	200
10 CONTINUE	—	100
STOP	—	1
END	—	—

Источник: Морган и Кэмпбелл (1973).

включить из анализа редко выполняемые операторы. Разумеется, этот анализ можно автоматизировать с помощью программ, подобных синтаксическому анализатору. Все эти усилия, по-видимому, можно оправдать в ряде практически интересных случаев, особенно тогда, когда есть автоматические, сравнительно недорогие измерители.

■

6.3.3. Сравнение компонентов системы

Выбор компонента системы — задача, возникающая весьма часто при проектировании конфигурации системы и иногда при попытках усовершенствования. Как обсуждается в гл. 8, проектирование конфигурации включает выбор доступных компонентов оборудования, например центральных процессоров, каналов, дисководов, лентопротяжных устройств, печатающих устройств, терминалов и контроллеров связи, которые предстоит соединить между собой. Иногда приходится выбирать даже компоненты программного обеспечения, например целые операционные системы, программы учета, драйверы ввода-вывода и связи и другие программы, часто рассматриваемые в качестве компонентов системы, например трансляторы, редакторы и системы обслуживания баз данных. Работы по усовершенствованию могут выявить желательность замены или добавления одного или нескольких компонентов системы. Затем новые компоненты должны быть выбраны среди нескольких вариантов, обычно имеющихся на рынке. Как в

случае проектирования, так и в случае усовершенствования, выбор иногда ограничивается одним из нескольких похожих изделий одного поставщика, а иногда включает компоненты, продаваемые различными изготовителями.

Наше обсуждение проблемы выбора компонентов в этой главе будет кратким. Дополнительные детали можно найти в гл. 7 и 8. Кроме того, аспекты производительности программ при выборе компонентов программного обеспечения рассматриваются в гл. 9.

В общем с точки зрения производительности выбор компонента требует, чтобы было известно влияние характеристик компонента на производительность системы. Если остальная часть системы и рабочая нагрузка полностью специфицированы, как в случае работ по усовершенствованию, n компонентов-кандидатов можно сравнить, анализируя производительности n конфигураций, соответствующих n возможным выборам. Этот анализ выполняется во времени измерения или, что часто гораздо удобнее, во время моделирования. Как обычно, решение о выборе будет основываться не только на производительности, но и на стоимости, надежности, легкости эксплуатации, простоте подсоединения и других (возможно, локальных и случайных) факторах. Подтверждение того, что компонент удовлетворяет определенным минимальным требованиям производительности, во многих случаях оказывается более важным для процесса выбора, чем результаты сравнительного исследования производительности. Иногда может оказаться необходимым анализ производительности только одной конфигурации, а именно той, которая содержит компонент-кандидат, оказавшийся наиболее желательным по результатам проверки по всем остальным факторам. Ситуация проектирования, определенно, сложнее, так как остальная часть системы вместе с рабочей нагрузкой может быть описана только схематично и неполно. Мы обсудим эту проблему в гл. 8.

Следует отметить, что поведение компонентов, особенно компонентов оборудования, часто можно удовлетворительно охарактеризовать параметрами, которые в отличие от используемых в этой книге индексов производительности системы не зависят от того, как используется данный компонент и при каких входных воздействиях. Это параметры того же типа, что сопротивление для линейного резистора или коэффициент усиления для усилителя. В качестве примера можно назвать время доступа к основной памяти, время оборота диска или барабана, время перехода с дорожки на дорожку и максимальное время поиска для диска с подвижными головками, скорость передачи данных для канала или устройства ввода-вывода, размер памяти, занятой компонентом программного обеспечения, и его максимальное и минимальное времена выполнения. Если при выборе компонента мы можем выразить желаемые характеристики его производительности в терминах значений параметров или их диапазонов, то процесс выбора, как при усовершенствовании, так и при проектировании может целиком основываться на документации, поставляемой изготовителями. Однако было бы все же полезно до окончательного решения удостовериться

экспериментально, что действительная производительность выбранного компонента совпадает с опубликованной. Эта процедура выбора не делает абсолютно бесполезной упомянутую выше работу по оценке. Единственное отличие состоит в том, что в этом случае цель работы заключается скорее в получении значений и диапазонов для параметров компонента, нежели в определении производительности одной или более конфигураций, включающих возможные компоненты.

Существуют по крайней мере два компонента системы, которые невозможно полностью охарактеризовать параметрами этого типа: центральные процессоры и операционные системы. Мы уже видели в разд. 3.3.4, как можно сравнивать скорости различных ЦП. При этом используются такие методы, как ручной подсчет (называемый *ручным хронометражем*) или измерение, а рабочая нагрузка центрального процессора характеризуется смесью команд или набором ядер. Трудной проблемой в сравнении центральных процессоров оказывается выбор классов команд, включаемых в смесь, и оценка частоты каждого из них. Другими словами, как получить модели одной и той же рабочей нагрузки для процессоров с различными наборами команд? Этот вопрос будет обсуждаться в разд. 6.5.

Сравнение различных операционных систем для одной конфигурации оборудования является важной проблемой. Симс (1974) экспериментально подтвердил интуитивное предположение: операционная система действительно может сильно изменить производительность. Он сравнил три диалоговые операционные системы: TSS/360, CP/67 и MTS, управляющие одной и той же системой IBM 360/67 в ряде конфигураций (один процессор или два процессора; один, два или три модуля основной памяти; диски IBM 2311 или 2314; с барабаном и без него). Сравнение операционных систем выдвигает в основном те же проблемы, что и сравнение предложений различных поставщиков, несколько упрощенные тем, что эти системы выполняются на одном и том же оборудовании. В качестве критериев сравнения следует выбрать те же самые индексы. Для диалоговых операционных систем, например, наиболее важным является время ответа, но пропускная способность также небезразлична. Возникают те же самые проблемы описания рабочей нагрузки, и обычно для их решения используются те же подходы. Некоторые проблемы описания рабочей нагрузки и их решения, описанные Симсом, мы рассмотрим в разд. 6.5. В терминах как времен ответа, так и пропускных способностей для искусственных рабочих нагрузок, созданных Симсом, система MTS показала на той же самой конфигурации оборудования приблизительно в 3 раза более высокую производительность, чем TSS. Системы CP и TSS обладают приблизительно одинаковой производительностью, когда CP работает с одним процессором, одним модулем памяти и медленными дисками, а TSS — с двумя процессорами, тремя модулями памяти и быстрыми дисками.

6.4. МЕТОДЫ ОЦЕНКИ ДЛЯ РАБОТ ПО ВЫБОРУ

Тот факт, что различные варианты, рассматриваемые в исследовании по выбору, обычно существуют и доступны, делает возможным сравнение их производительностей методами измерения. Разумеется, можно использовать и другие методы, но метод измерений удобен почти во всех случаях и, следовательно, наиболее широко распространен.

При выборе поставщика измерения, обеспечивающие основу для обоснования предложений и сравнения производительностей, обычно выполняются каждым поставщиком. В принципе будущие потребители могут оценивать производительности различных систем посредством аналитического или имитационного моделирования. Построить столько моделей, сколько предложений предстоит проверить, по всей вероятности, недопустимо дорого. Можно использовать коммерческие моделирующие программы (см. разд. 3.1), и это действительно делалось в ряде работ по выбору. Однако точность результатов моделирования может оказаться в некоторых случаях слишком низкой как для обоснования предположения, так и для сравнения. Аналитические методы, по-видимому, недостаточно разработаны для обеспечения необходимой степени детальности и, следовательно, требуемой точности, особенно для сравнительных исследований. Изготовители для получения нужных данных могут использовать обычные измерители. Важным измерителем оказывается секундомер или подобный ему прибор для измерения израсходованного времени, а также времени прохождения и ответа. Другие, быть может, более сложные, измерители понадобятся для наблюдения вторичных индексов производительности во время управляемых экспериментов. Тот факт, что данные о производительности для данного исследования получены разными изготовителями с помощью различных методов и измерителей, может внести в результат систематические ошибки неизвестного знака и величины. Ясное и однозначное определение индексов и операционных процедур (см. пример 6.2) несомненно уменьшит эти ошибки. Просьба к изготовителям подробно описать использованные измерители и процедуры сбора и сжатия данных также будет способствовать повышению доверия к результатам.

Большая часть приведенных выше доводов относится и к выбору установки. В этих исследованиях методы измерения также являются наиболее удобными из-за того, что они предоставляют пользователю возможность оценить такие важные аспекты разных установок, как легкость применения, доступность и программные вспомогательные средства. Ни моделирование, ни аналитические методы не подходят для этого типа выбора. Нужные измерители обычно очень просты. В большинстве случаев годится любой измеритель времени, соответствующий тем величинам времени, которые подлежат измерению.

Выбор компонентов — единственная область, в которой моделирование может в действительности заменить измерение. Пока система не существует, как, например, при проектировании конфигурации,

имитационное моделирование и в самом деле будет полезно, хотя его можно избежать в простейших случаях, используя аналитические модели. При выборе центрального процессора смеси команд и ядра можно оценить вручную (это тоже форма моделирования) или выполнить. При сравнении операционных систем по всей вероятности более удобны измерения, если они возможны.

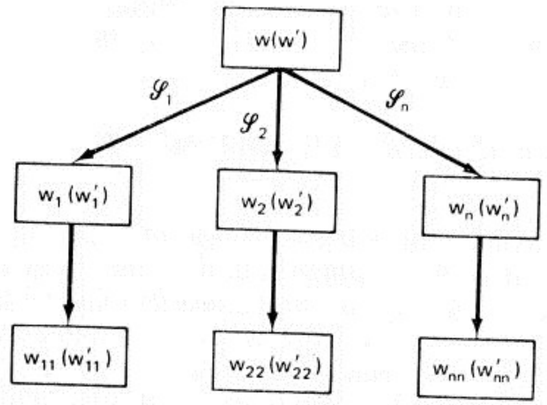
6.5. ОПИСАНИЕ РАБОЧЕЙ НАГРУЗКИ ДЛЯ РАБОТ ПО ВЫБОРУ

В предыдущих разделах этой главы был выдвинут ряд вопросов, относящихся к описанию рабочей нагрузки. В этом разделе мы коснемся двух проблем, особенно важных при выборе поставщика и сравнении производительности систем: проблемы системно-независимого описания и проблемы *прогнозирования рабочей нагрузки*.

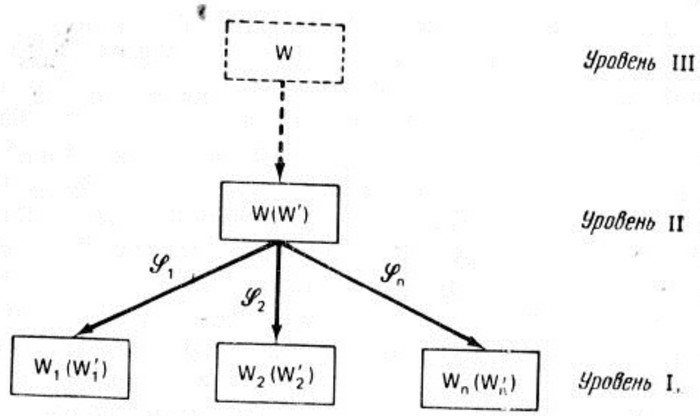
Необходимость системно-независимых моделей рабочей нагрузки и возможные принципы их построения обсуждались как в гл. 5 (особенно в разд. 5.2.3), так и в данной главе. Однако крайняя важность этого вопроса в задачах выбора требует единого рассмотрения, включая иллюстрацию на нескольких специфических примерах.

В общем виде проблема может быть описана следующим образом. Пусть даны n предложенных систем $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$ и рабочая нагрузка W (которая является, например, производственной рабочей нагрузкой текущей установки), сконструируем n исполняемых искусственных моделей рабочей нагрузки $W_1', W_2', \ldots, W_n'$, таких, что W_i' эквивалентна W по отношению к системе $\mathcal{S}_i$ и индексам производительности, выбранным в качестве критериев сравнения. Разумеется, это утверждение имело бы смысл, только если бы было определено значение эквивалентности между W_i' и W. W — рабочая нагрузка установленной в текущий момент системы $\mathcal{S}$. Можно ли перенести W на $\mathcal{S}_i$ и как? Если существуют нетривиальные отличия $\mathcal{S}_i$ от $\mathcal{S}$, любая попытка определить эквивалентность в терминах физических (аппаратных или программных) ресурсов обречена на неудачу. Только описание W в терминах того, что мы назвали логическими ресурсами в разд. 5.2.3, может позволить нам перенести W на другую систему. В идеальном случае такое системно-независимое описание должно основываться на задачах обработки информации, которые должны выполняться на установке, или, иными словами, на нуждах пользователей. Обозначим это идеальное описание через w. Если бы наше знание коллектива пользователей позволило нам определить воздействие каждой предлагаемой системы на поведение пользователей, мы смогли бы преобразовать w в рабочие нагрузки $w_1, w_2, \ldots, w_n$, эквивалентные задачам обработки информации соответственно на системах $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$. Эти рабочие нагрузки характеризовались бы своими запросами в терминах ресурсов языка высокого уровня, как показано на рис.6.7(а). Эти запросы могут быть преобразованы в запросы ресурсов системы компиляторами, работающими в каждой системе. Однако наши зна-

ния в этой области еще далеко не достаточны (см. обсуждение этой проблемы в разд. 5.2.3). Таким образом, нам обычно приходится принимать, что выходная информация коллектива пользователей не зави-



(a)



(b)

Рис. 6.7. Деревья рабочей нагрузки для задач выбора поставщика: (a) — идеальное решение; (b) — практическое решение. Символ, представляющий модель рабочей нагрузки, следует в скобках за символом самой рабочей нагрузки. Рабочие нагрузки и их модели обозначены строчными символами в (a) и прописными в (b). Рабочая нагрузка характеризуется в терминах физических ресурсов на уровне I, логических ресурсов на уровне II и запросов на обработку информации на уровне III. Язык является машинным языком на уровне I, языком высокого уровня на уровне II и языком описания задачи на уровне III. Компиляторы являются преобразователями между уровнями I и II, а разработчики программы и кодировщики — преобразователями между уровнями II и III.

сит от различных вариантов. Следовательно, ситуация будет такова, как изображено на рис. 6.7 (b), где W описано в терминах запросов ресурсов на языке высокого уровня. Иначе говоря, W считается набором программ на языке высокого уровня и необходимых для них файлов.

Мы будем предполагать, что независимо от результата процедуры выбора язык, используемый программистами для написания данной задачи, не изменится. Заметим, что мы легко могли бы добавить к диаграммам на рис. 6.7 дополнительные уровни, введя различие между операционной системой и уровнем оборудования (которые оба отнесены к уровню I) или представляя, например, виртуальные машины. Однако мы сознательно оставляем диаграммы по возможности простыми.

При предположениях, графически представленных на рис. 6.7 (b), описание W в терминах машинных ресурсов на языке высокого уров-

Таблица 6.13

Характеристики идеальных эталонов и процедур, оперирующих эталонами

1. Написаны для стандартной версии (если она существует) языка высокого уровня, чтобы минимизировать преобразования текста программы.
2. Отлажены настолько, что результаты предсказуемы на всех сравниваемых машинах.
3. Файлы данных должны легко восприниматься всеми системами и, кроме того, должны быть согласованы с архитектурой каждой из систем, чтобы избежать неправильных сравнений или потери значимости результатов.
4. Преобразование эталонов поставщиками, контролируемое, как и в случае управляемой среды, для получения информации о легкости преобразования всей рабочей нагрузки.
5. Автоматическая проверка результатов работы эталонов.

Источник: Бейрд и Джонсон (1974).

ня не зависит от системы. Поэтому исполняемая модель W' нагрузки W, состоящая из набора эталонов (см. пример 6.2) или из синтетических заданий, основанных скорее на логических, чем на физических ресурсах (см. пример 6.3), могла бы использоваться в качестве общей рабочей нагрузки в работе по сравнению производительностей. На практике программы такого набора обычно нельзя переносить без изменений из-за различий между внутренними представлениями данных, между внутренними соглашениями операционных систем и между версиями одного и того же языка в разных системах. Проблема адаптации эталонов или синтетических заданий к системе, которая должна решаться каждым поставщиком в процедуре выбора, описанной в разд. 6.2.1, часто оказывается нетривиальной. Бейрд и Джонсон (1974) перечислили характеристики идеального набора эталонов, а мы собрали их в табл. 6.13. Автоматические вспомогательные средства преобразования эталонов (задача весьма дорогая и расходующая много времени) предлагались Бейрдом и Джонсоном (1974) и другими авторами.

Эти доводы, оправдывающие использования синтетических заданий с заказом ресурсов на языке высокого уровня при выборе систем, основывались на предположении, что существует набор языков высо-

кого уровня для всех сравниваемых систем. Модель W' рабочей нагрузки W [см. рис. 6.7(b)] будет состоять из программ, написанных на языках из этого набора. Существует по крайней мере два важных случая, в которых это предположение не удовлетворяется: случаи сравнения ЦП и сравнения диалоговых систем. Мы проиллюстрируем их следующими двумя примерами.

Пример 6.4. В примере 5.5 мы показали, как можно сравнивать скорости двух центральных процессоров A и B с использованием смеси команд и времен выполнения каждой отдельной команды. Зададимся вопросом, представляет ли данная смесь команд

Таблица 6.14

Степень пригодности машинных языков шести ЦП для представления внутреннего цикла программы на АЛГОЛе

Программа

```
e:=p;
loop:if e≠0 then
begin f:=(t and area (e+1));
    if f≠0 then area (e+2):=area (e+2)+s;
    e:=area (e); goto loop;
end;
```

Машина	Число использованных команд	
	Поставщик	Керниган и др.
DEC PDP-10	4	4
Honeywell — Bull G625	7	5
IBM 360/370	7	5[1]
Xerox SIGMA 5	6	5
RC 4000	7	7[2]
UNIVAC 1106	7	6

Источник: Керниган и др. (1972).
[1]) Это решение может быть медленнее некоторых семикомандных решений.
[2]) Скорость можно увеличить, используя девять команд.

(например, смесь Флинна в табл. 5.5) одинаковую рабочую нагрузку для A и B? В общем случае ответ должен быть отрицательным. Идентичные задачи обработки информации обычно будут кодироваться для разных ЦП в программы в машинном виде с различными долями команд из разных классов. Одинаковые программы на языке высокого уровня будут транслироваться двумя компиляторами в программы на внутреннем языке машины с различными частотами команд одного и того же класса. То, что различные архитектуры дают различные смеси, ясно видно на примере двух смесей из табл. 5.5, если предположить, что эти две моделируемые рабочие нагрузки не слишком отличаются друг от друга статистически. Значение понятия класса команд само по себе сомнительно. Например, должны ли мы отнести сложения «регистр — память» (сложить содержимое регистра R1 с содержимым ячейки памяти X и поместить результат в R1) к тому же классу (т. е. к классу операций с фиксированной точкой), что и сложения «регистр — регистр» (сложить содержимое регистра R1 с со-

держимым регистра R2 и поместить результат в R1)? Или первую из этих операций следует отнести к классу «считать/записать»? Или к обоим классам?

Единственно разумное решение, пока мы интересуемся только сравнением в первом приближении, это приготовить две различные смеси команд, одну для машины A, другую для машины B, из одной и то же рабочей нагрузки на языке высокого уровня или из модели этой нагрузки (например, из набора ядер). Если две машины A и B — такие, что приведены в табл. 5.6, и смесь для машины A — смесь Гибсона, а смесь для B — смесь Флинна, то A приблизительно на 26% медленнее, чем B. Чтобы сделать рабочую нагрузку более реалистичной с точки зрения ЦП, следует включить в нее все программы, которые выполнял бы тот или иной ЦП, если бы его выбрали. Эта рабочая нагрузка, вообще говоря, должна содержать операционную систему, компиляторы и все программное обеспечение системы, не выполняемое процессорами ввода-вывода. Однако, к сожалению, это легче провозгласить, чем выполнить практически. Например, системные программы могут быть недоступны в момент сравнения или интенсивность их использования рабочей нагрузкой пользователя может быть трудно предсказать.

При выборе ЦП часто руководствуются несколькими дополнительными критериями (кроме скорости), например потенциальной точностью (которая связана непосредственно с длиной слова) или пригодностью набора операций для планируемых применений системы. Приблизительной мерой этой пригодности может быть минимальное число машинных команд, необходимых для записи определенных задач, которые считаются удовлетворительно представляющими применение. Однако времена выполнения этих задач также важны; иначе машины с длинными управляющими словами получат преимущество перед остальными. Возможно, наилучшим критерием было бы время программирования одной и той же задачи для разных машин, но его трудно измерить. Заметим, что определение минимально необходимого количества команд исключило бы влияние программистов, а также фаз генерации кодов различными компиляторами. Однако определить минимум очень трудно даже для очень простой программы, как показано в табл. 6.14, в которой приведены «минимальные» числа, представленные шестью поставщиками, и соответствующие числа, полученные Керниганом и др. (1972) для внутреннего цикла простой программы на АЛГОЛе. Эта таблица иллюстрирует на предельно простом случае описанную нами методику сравнения ЦП и их наборов команд. Мы предполагаем, что эта методика, основанная на модели рабочей нагрузки на языке высокого уровня, концептуально корректна. Однако эта таблица показывает, что результатам, полученным другими, включая поставщиков, нельзя полностью доверять. Возможно, что результатам поставщиков можно доверять больше, чем мы склонны это делать, так как предположение, что изготовители знают свои машины много лучше, чем средний пользователь, не всегда верно. ■

В разд. 6.3.3 мы обсуждали аспекты оценки производительности для процедуры выбора изготовителя (поставщика), обращая, в частности, внимание на установку с пакетной обработкой. Там, а также в начале этого раздела мы увидели, что модели, составленные из этапов или синтетических заданий на языке высокого уровня, являются наилучшими из известных способов представления рабочей нагрузки в этих случаях. Какие модели рабочей нагрузки следует использовать при сравнении диалоговых систем?

Выбрать рабочую нагрузку W на уровне II, как показано на рис. 6.7(b), почти всегда невозможно в случае диалоговых систем из-за неизбежных и значительных различий, обычно существующих между диалоговыми командными языками. Сценарий, подготовленный для системы A на ее командном языке, невыполним в системе B. Здесь необходимо принять подход, в некотором смысле близкий к подходу,

представленному на рис. 6.7(а). Мы должны специфицировать сценарии на уровне задач обработки информации (это модель w') и закодировать их в стольких различных версиях, сколько у нас систем (модели $w'_1, \ldots, w'_n$). Примером спецификации сценария на уровне III может служить приведенный в табл. 5.9. Для различных систем эта спецификация сценария будет транслироваться в полностью различные последовательности команд. Модели рабочей нагрузки $w'_1, \ldots, w'_n$ будут также содержать механизмы имитации времен обдумывания, если сценарии не будут подчинены пользователям по ходу экспериментов. Эти сценарии будут инициировать выполнение различных программ, выбранных так, чтобы представлять различные способы использования: редакторы, компиляторы, отладчики, прикладные программы. Прикладные программы, которые должны компилироваться или отлаживаться, файлы, которые должны редактироваться, могут использоваться разработчиком модели рабочей нагрузки, как если бы они были эталонами или синтетическими заданиями (и их данными) для сравнения пакетных систем. В любом случае их проектирование или выбор должны основываться не на запросах физических ресурсов, а на запросах в терминах ресурсов языка высокого уровня. Следующий пример описывает в некоторых деталях одну из экспериментальных схем, использованную Симсом (1974) в его сравнении диалоговых операционных систем (см. также разд. 6.3.5).

Пример 6.5. Предстоит сравнить системы TSS/360, CP/67 и MTS, работающие на IBM 360/67 с различной конфигурацией. Число активных терминалов поддерживалось относительно постоянным во всех сеансах измерения и равным приблизительно 24. Каждому терминалу был назначен сценарий на каждый сеанс. Нагрузка изменялась от сеанса к сеансу путем изменения пропорции сценариев, назначенных

Таблица 6.15

Сценарии, использованные для моделирования рабочей нагрузки при сравнении диалоговых систем

Номер сценария	Название	Описание
1	FORTRAN	Вызывает компиляцию программы на ФОРТРАНе из 75 карт
2	PLILG	Вызывает компиляцию программы на PL/I из 434 карт
3	PLISM	Вызывает компиляцию программы на PL/I из 47 карт
4	FORTEX	Вызывает выполнение синтетического задания, связанного с вычислениями (оно производит 1 миллион сложений и печатает строку на терминале)
5	PAGE	Вызывает выполнение программы, которая использует большой массив и обращается к новой странице при каждой выполняемой операции
6	EDIT	Выполняет несколько функций редактирования (находит строку в программе, передвигает указатель вверх или вниз, печатает некоторую информацию и т. д.)

Источник: Симс (1974).

терминалам. Было подготовлено шесть сценариев: три типа компиляции (два для PL/I и один для ФОРТРАНа), два типа выполнения (один, связанный с вычислениями, а другой — с постраничным вводом-выводом) и один типа редактирования. Эти сценарии и их основные характеристики перечислены в табл. 6.15. Изменениями нагрузки управляли посредством изменения соотношения между числом терминалов, выполняющих редактирующий сценарий, и числом терминалов, выполняющих компилирующий и сценарий выполнения. Это соотношение менялось от 1 : 5 до 5 : 1. Симс считал, что нормальная рабочая нагрузка соответствовала отношению приблизительно 2 : 1.

Интересно отметить, что Симсу удалось выразить *фактор нагрузки* каждой системы (см. разд. 5.2.2) как функцию числа терминалов, выполняющих каждый сценарий. Например, если мы обозначим через n_i число терминалов, выполняющих i-й сценарий из табл. 6.15, то фактор нагрузки для CP/67 равен

$$L_{CP} = 4n_1 + 8n_2 + 6n_3 + 4n_4 + 24n_5 + 2n_6, \tag{6.22}$$

а для MTS —

$$L_{MTS} = n_1 + a_2 n_2 + 6n_3 + a_4 n_4 + 6n_5 + n_6$$
$$(a_i = 8, \text{ если } n_i \leqslant 4;\ a_i = 2.5, \text{ если } n_i > 4;\ i = 2, 4). \tag{6.23}$$

Таким образом, комбинации пар (терминал, сценарий), перечисленные в табл. 6.16, дают приблизительно одинаковую производительность под управлением

Таблица 6.16

Рабочие нагрузки, дающие приблизительно одинаковую производительность в экспериментах, описанных в примере 6.5, на системе CP/67

Номер сценария (см. табл. 6.15)	Число терминалов, работающих по каждому сценарию			
	Нагрузка a	Нагрузка b	Нагрузка c	Нагрузка d
1	1	—	3	10
2	3	—	5	—
3	5	1	2	—
4	6	2	4	5
5	—	2	—	1
6	9	19	10	8
Всего терминалов	24	24	24	24
L_{CP}	100	100	100	100
L_{MTS}	79	53	69.5	36.5

системы CP/67. В таблице также показаны соответствующие значения L_{MTS}. Как и следовало ожидать, попытка скомбинировать различные факторы нагрузки в один, чтобы получить общий показатель, по которому можно было бы планировать и сравнивать индексы производительности, была безуспешной (см. задачу 6.16), хотя и не бесполезной. Две рабочие нагрузки, для которых данная система демонстрирует одинаковую производительность, вообще говоря, обрабатываются в другой системе с разными производительностями. Как отмечается в разд. 5.2.2, не существует сохраняющей эквивалентности рабочих нагрузок при переходе с одной системы на другую. Единственно правильными оказываются сравнения, в которых системы управляются моделями в точности одинаковой рабочей нагрузки.

Теперь кратко обсудим проблему прогнозирования рабочей нагрузки на существующей установке. Мы видели в разд. 6.3.3, что в работах по выбору часто оцениваются будущие рабочие нагрузки. При этом возникает важный вопрос: «Окажется ли система способной справиться с возросшей рабочей нагрузкой, которую создадут пользователи установки, скажем, через 2 года после ввода выбранной системы в эксплуатацию, и при каких затратах?» Разумеется, подобные проблемы возникают также и при исследованиях повышения производительности и при проектировании конфигурации.

Чтобы предсказать, какова будет рабочая нагрузка в будущем, мы можем использовать методы, основанные на экстраполяции тенденций. История рабочей нагрузки на данный момент на данной установке дает скорости роста, по которым вычисляются ожидаемые в ближайшем будущем рабочие нагрузки. При этом предполагается, что существующие тенденции сохранятся, но, кроме того, вносятся коррекции, если становится известна дополнительная информация о новых применениях, об изменениях в коллективе пользователей, о новом в экономической ситуации или о других нарушающих факторах (например, возможное создание другой установки в той же организации). Разумеется, неточность прогноза, определяемая, например, доверительными интервалами оценок параметров рабочей нагрузки, будет увеличиваться по мере удаления в будущее. По мере того как мы пытаемся отойти все дальше и дальше от настоящего, доверительные интервалы расширяются, а вероятность непредсказуемых нарушающих влияний становится больше. За определенным пределом любую оценку следует считать бессмысленной. Обычно считается, что для типичных вычислительных установок предсказания не должны распространяться далее, чем на 5 лет, а иногда и на меньший период.

Какие параметры рабочей нагрузки следует экстраполировать? Вообще говоря, разумно обратиться к описанию рабочей нагрузки, выбранной для проводимого исследования. Если это описание состоит из количественных параметров (например, запрос времени ЦП, запрос памяти, общее машинное время за месяц, и т. д.), мы можем продолжить в будущее тенденции, известные из данных за прошедшее время и получить таким способом модели будущих рабочих нагрузок, однотипных с используемыми. Экстраполяции запросов на физические ресурсы по учетной информации должно быть достаточно для определения, например, ожидаемой даты насыщения существующей емкости системы. В задачах выбора, когда существующая система должна быть заменена одной из сравниваемых, модели рабочей нагрузки, использующие физические ресурсы, оказываются неподходящими. Таким образом, модель, построенная на уровне II или на уровне III (см. рис. 6.7), могла бы использоваться для целей прогнозирования. При отсутствии каких-либо указаний мы можем предположить, что типы прикладных программ существенно не изменятся. Таким образом, изменится только общее количество поступающих заданий за определенный период времени. Другими словами, задачи не изменятся, изменится только

частота поступления программ, выполняющих их. Использование диаграмм, подобных приведенной на рис. 6.8, для целей прогнозирования рабочей нагрузки основывается на этом предположении. Заметим, что для диалоговых установок эквивалентное предположение состоит в том, что будет меняться только число активных терминалов, а не природа и не относительные доли видов деятельности. Настройка исполняемой модели рабочей нагрузки или изменение способа ее использования

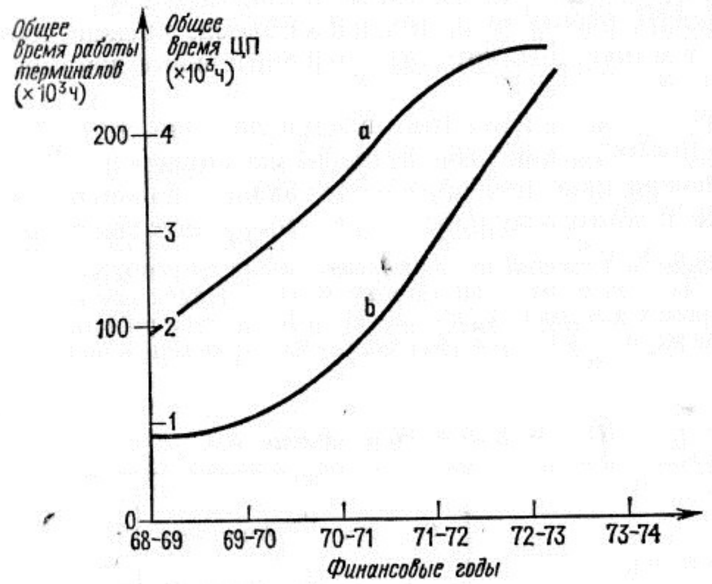


Рис. 6.8. Прогноз рабочей нагрузки Дартмутской системы разделения времени, создаваемой «своими» пользователями: (a) — общее время работы терминалов; (b) — общее время ЦП. (*Источники*: Кивитский вычислительный центр, Дартмутский колледж, отчеты за период 1969—1971 и 1971—1973 гг.)

при такой модификации просты. Если ожидаются изменения типов прикладных программ, их введение в такую модель может оказаться существенно не сложнее, чем изменение частоты поступления программ. Например, если модель была сконструирована с помощью методов, подобных методам Ac и Ad из разд. 5.3.5, то легко изменить соотношения между количествами моделей заданий, представляющих различные классы заданий. Если модель рабочей нагрузки составлена из синтетических заданий типа описанных в примере 6.3 (см. табл. 6.12), то относительно просто учесть ожидаемые изменения в прикладных программах путем изменения долей различных ресурсов языка высокого уровня, запрошенных моделью (это запросы ресурсов, перечисленные в табл. 6.11, предпоследний столбец). Эти изменения могут основываться не только на догадках, но и на экстраполяциях, если сохраняются записи истории этих запросов, хотя бы за недавнее прошлое. К сожалению, на установке обычно не проводятся измерения подобного рода, и поэтому эффекты непрерывного изменения прикладных программ при прогнозировании рабочей нагрузки либо игнорируются,

либо о них только догадываются. Кроме того, отсутствие накопленных данных в этой области не позволяет оценить воздействие внезапных изменений. Одним из очень важных и неизбежных изменений такого типа является, конечно, установка новой системы. Какое воздействие окажут характеристики и производительность новой системы на кратковременное и долговременное поведение коллектива пользователей? Как в свою очередь это поведение будет воздействовать на природу и интенсивность рабочей нагрузки? Как отмечалось в разд. 5.1, необходимо проделать работу значительного объема, прежде чем на эти вопросы удастся получить хотя бы предварительные документированные ответы.

Мы теперь покажем вкратце, следуя подходу, предложенному Гудом и Муном (1972), как рабочая нагрузка университетского вычислительного центра пакетной обработки заданий может проектироваться на будущее в контексте работы по выбору поставщика.

Пример 6.6. Университет рассматривает замену системы, установленной в вычислительном центре, на более новую и мощную. Действующая установка обеспечивает только режим пакетной обработки. На первом этапе работы сделано предположение, что так же будет и в новой системе. Поскольку новая система почти наверняка

Таблица 6.17

Данные для примера 6.6

Факультет, i	Студенты		Аспиранты		Показатель годового индивидуального использования существующей системы (часы ЦП/усл. асп.)	
	$u_i(0)$	U_i	$g_i(0)$	G_i	$h_i(0)$	r_i (%)
Точных наук	1840	40	316	10	2.71	5
Технических наук	1320	35	282	15	10.24	8
Искусств и общественных наук	4240	65	564	25	0.61	7
Агрономии	760	20	156	5	2.46	4
Управления	1400	45	268	20	3.38	7
Медицинский	720	15	128	5	5.59	5

будет обеспечивать диалоговую обработку; следует ожидать, что значительная доля существующей рабочей нагрузки (создаваемая пользователями, пишущими и отлаживающими программы) станет диалоговой. Кроме того, возможно, при наличии новой системы будет стимулирован ряд новых применений, например подготовка документов, информационный поиск, проектирование с помощью ЭВМ, что внесет свой вклад в увеличение общей рабочей нагрузки. Предсказание результатов этих изменений рабочей нагрузки, однако, является целью следующего этапа исследования, который мы не будем рассматривать в этом примере.

Таблица 6.17 показывает состав каждого из факультетов университета и средний рост числа студентов и аспирантов, прогнозируемый на 5 академических лет. Таблица также содержит показатель годового индивидуального использования существующей системы для каждого факультета и скорость его ожидаемого ежегодного прироста. Показатель индивидуального использования системы для факультета вычислялся де-

лением общего времени ЦП существующей системы, предоставленного этому факультету, на число *условных аспирантов*. Этот подход основывался на наблюдении, что вычислительные запросы (в терминах процессорного времени) 40 студентов приблизительно равны запросу одного аспиранта, а использование компьютера сотрудниками факультета можно без существенной потери точности рассматривать так, как если бы они были аспирантами [см. Гуд и Мун (1972)].

Каково должно быть соотношение между емкостями существующей и новой систем, если новая система должна обрабатывать без задержек рабочую нагрузку, ожидающуюся к концу рассматриваемого 5-летнего периода? Эту рабочую нагрузку, выраженную в терминах процессорного времени действующей системы, можно оценить по формуле

$$H(t) = \sum_{i=1}^{6} \mathrm{egs}_i(t) \cdot h_i(t), \tag{6.24}$$

где t — время в годах (начальное значение $t=0$ соответствует текущему академическому году), h_i — ежегодная потребность факультета i в процессорном времени на одного условного аспиранта, egs_i — число условных аспирантов на факультете i, а H — общая ежегодная потребность в процессорном времени.

Значения $\mathrm{egs}_i(t)$ и $h_i(t)$ можно вычислить с помощью следующих выражений:

$$\mathrm{egs}_i(t) = g_i(t) + \frac{u_i(t)}{40} = g_i(0) + tG_i + \frac{u_i(0) + tU_i}{40}, \tag{6.25}$$

$$h_i(t) = h_i(0)\,(1 + r_i)^t, \tag{6.26}$$

где g_i и u_i — числа аспирантов и студентов соответственно на факультете i; G_i и U_i — планируемый средний прирост числа аспирантов и студентов факультета i; r_i — ожидаемый прирост годовой потребности в процессорном времени типичного аспиранта факультета i. Значения величин, входящих в правую часть равенств (6.25) и (6.26), приведены в табл. 6.17. Таблица 6.18 показывает результирующие значения $u_i(5)$,

Таблица 6.18

Прогноз на 5 лет состава факультетов и использования компьютера для примера 6.6

Факультет, i	Студенты $u_i(5)$	Аспиранты $g_i(5)$	Условные аспиранты $\mathrm{egs}_i(5)$	Ежегодное индивидуальное использование существующей системы (часы ЦП/egs) $h_i(5)$
Точных наук	2040	366	417	3.46
Технических наук	1495	357	394	15.04
Искусств и общественных наук	4565	689	803	0.85
Агрономии	860	181	202	2.99
Управления	1625	368	408	4.74
Медицинский	795	153	173	7.13

$g_i(5)$, $\mathrm{egs}_i(5)$, $h_i(5)$. Применяя (6.24) к результатам в этой таблице, получим $H(5) = 11\,822$ часа процессорного времени. Так как полностью загруженная машина вырабатывает около 600 часов ежемесячно, то, считая, что нагрузка должна обрабатываться в течение приблизительно 10 месяцев (мы игнорируем гораздо меньшую летнюю нагрузку за 2 месяца), получим, что требуется система с вдвое большей вычислительной мощностью. Разумеется, если необходимо также, чтобы новая машина обрабатывала пиковую дневную нагрузку (без перераспределения ее на следующий день), и если мы предположим, что максимальные дневные нагрузки не будут превышать среднюю

величину более чем вдвое, то нужно приобретать систему с мощностью, в четыре раза большей, чем у существующей системы.

В этом примере рабочая нагрузка установки описывалась общей потребностью в процессорном времени для заданий, которые предстоит обработать в течение академического года. При использовании этой характеристики только для целей прогнозирования мы сделали неявное допущение, что природа рабочей нагрузки существенно не изменится в течение последующих 5 лет, так что соотношение между объемами работы ввода-вывода и центрального процессора приблизительно сохранится прежним. Кроме того, когда мы пришли к выводу о требуемой мощности центрального процессора новой машины, мы сделали неявное предположение, что архитектура новой системы будет в достаточной степени «подобна» структуре существующей системы, так что все еще можно будет пренебрегать влиянием процедур ввода-вывода. Иными словами, мы предположили, что существующая система лимитируется мощностью ЦП и что это будет иметь место и для новой системы. Простые вычисления, основанные на табл. 6.17, показывают, что первое предположение верно. Если бы последнее предположение не удовлетворялось, рабочую нагрузку следовало бы описывать иначе, чтобы избежать принятия в корне неверных решений. Относительные влияния различных ресурсов производительности системы обсуждаются более подробно в гл. 7 ■

Задачи

6.1. Предложите три индекса, которые, по вашему мнению, больше подходят (чем перечисленные в табл. 6.2) в качестве критериев сравнения, показанных в той же таблице под заголовком «класс». Как вы бы измеряли каждый из этих трех индексов в эксперименте по сравнению пакетного и диалогового режимов обработки?

6.2. Для изучения темпа технического прогресса в области вычислительных машин вы можете выбрать некоторое значение мощности по переработке информации,

Таблица 3.6.1

Стоимости некоторых мини-компьютеров, обладающих приблизительно одинаковой мощностью

Мини-компьютер	Дата выпуска	Стоимость (долл./ч)
P ≅ 16 000 операций/с		
Hewlett-Packard 2114C	Январь 1968	2.12
TEC Model 520 PCP	Июнь 1969	1.27
Electronic Processors EPI-118	Ноябрь 1970	0.86
Data General NOVA 1200	Декабрь 1971	0.77
P ≅ 24 000 операций/с		
Honeywell DDP-516	Октябрь 1966	3.46
Interdata Model 4	Июнь 1968	2.30
General Automation SPC-16	Май 1970	1.41
Digital Computer Control D-216	Август 1970	1.17

Источник: Червены и Найт (1973).

определенной, например, по Найту, и проследить эволюцию стоимостей машин приблизительно одинаковой мощности на протяжении ряда лет. В табл. 3.6.1 перечислены цены и даты выпуска некоторых мини-компьютеров одинаковой мощности для двух значений мощности. Используйте данные в таблице для предположений об изме-

нениях во времени g и K в уравнении (6.11), полагая, что эти два параметра зависят только от времени, а не от P или C.

6.3. Следуя упрощенному подходу, описанному в примере 6.1, определите, будет ли меняться значение N (число машин, которые следует установить) при минимальной стоимости C по сравнению с изображенным на рис. 6.3, если задать одно из следующих изменений:

(a) общая мощность P удвоена ($P=2\times10^6$ команд/с);

(b) вероятность отказа каждой системы уменьшена в 10 раз ($p_f=0.01$);

(c) площадь географического района, в котором размещены пользователи, уменьшена в 4 раза, т. е. среднее расстояние уменьшено в 2 раза.

6.4. Выведите выражение для общей стоимости системы, рассмотренной в примере 6.1, предполагая, что

(a) стоимость прерываний обслуживания пропорциональна доле неиспользуемого оборудования; таким образом, если $N=2$, то $C_i=K_i\left(\frac{1}{2}p_f+\frac{2}{2}p_f^2\right)$; и

(b) вероятность отказа устройства пропорциональна его мощности.

Если все параметры имеют те же значения, что и в примере 6.1, а $p_f(N=1)=0.1$, то каково значение N, соответствующее минимальной общей стоимости?

6.5. Нарисуйте график зависимости общей мощности по переработке информации (в командах в секунду), достижимой при значениях N от 1 до 8, для заданной общей стоимости C. Воспользуйтесь подходом и численными значениями параметров, использованными в примере 6.1. Предположите стоимость равной минимальному значению на рис. 6.3.

6.6. Назовите причины (или ситуации), по которым при проектировании набора эталонов или синтетических заданий для задачи выбора поставщика отдается предпочтение методам, перечисленным в табл. 5.10, для выбора моделей заданий, которые должны быть включены в выполняемую модель рабочей нагрузки.

6.7. Сочтете ли вы попытку создания набора стандартных моделей рабочей нагрузки для использования при выборе поставщика-стоящей? Реалистичной? Полезной? Необходимой? Почему? Как вы относитесь к созданию большой библиотеки стандартных синтетических заданий, из которых любой покупатель может собрать частную модель рабочей нагрузки посредством выбора заданий и указания количества их копий и задания их аргументов?

6.8. Вам предстоит разработать набор эталонов для выбора одной системы из ряда систем, предложенных различными поставщиками. Новая система должна заменить существующую, работающую в пакетном режиме. Модель рабочей нагрузки, которую нужно разработать, должна представлять производственную рабочую нагрузку существующей установки. Предположим, что вы решили использовать метод типа Ac (см. табл. 5.10) для выбора моделей заданий. Будете ли вы делить задания в производственной рабочей нагрузке по

(a) языкам, на которых они написаны?
(b) расходу процессорного времени?
(c) расходу времени на ввод-вывод?
(d) отношениям времен ЦП и ввода-вывода?
(e) отношениям времен завершения заданий к их полным временам выполнения?
(f) их внешним приоритетам?
(g) каким-нибудь другим характеристикам?

Заметим, что классы заданий могут определяться более чем одним из приведенных выше параметров и можно привести доводы в пользу использования любого из этих параметров. Таким образом, вам следует обсудить эти аргументы и упорядочить параметры по вашему выбору. Разумно ли, на ваш взгляд, классифицировать задания или шаги заданий? Почему создатели моделей рабочей нагрузки уделяют столько внимания различию между этапами компиляции и выполнения (см., например, пример 6.2)?

6.9. Нужно сравнить производительности двух диалоговых систем A и B. Индекс, выбранный для сравнения,— среднее время ответа. Спланируйте эксперимент, который позволит вам провести это сравнение. Перечислите возможные источники

изменения и укажите, как бы вы экспериментально контролировали те из них, которые не должны вносить вклад в различие средних времен ответа. Укажите, какие статистические тесты вы бы выполнили, чтобы увидеть, существенно ли это различие, и как можно было бы проанализировать дисперсию этой разности.

6.10. Какие факторы обусловливают отличие между элементами второй строки и какие между элементами четвертой в табл. 6.8?

6.11. В некоторых операционных системах, например в GCOS для машин серии Honeywell 600, использование всех ресурсов заданием учитывается в терминах условных секунд процессорного времени. Например, заданию, которое требует менее 16K слов основной памяти, выписывают счет за количество процессорного времени, равное израсходованному в действительности, а заданию, требующему от 16K до 32K слов основной памяти, предъявляют счет за количество процессорного времени вдвое больше израсходованного. Исходя из предположения, что одна из сравниваемых установок в примере 6.3 работает под управлением GCOS, как бы вы изменили версию модели из табл. 6.12, чтобы ее можно было выполнять на данной установке?

6.12. У вас есть пакетная задача обработки информации, требующая определенной мощности. Эту мощность можно получить, либо приобретя небольшую систему, либо сняв в аренду (или купив) удаленную станцию ввода заданий и получая машинное время на большой установке. Приведите доводы в пользу и против каждого из вариантов и обсудите подробно метод, с помощью которого вы бы сравнили эти два решения с точки зрения производительности.

6.13. Нужно сравнить производительности двух мультипрограммных операционных систем пакетной обработки, работающих на одной и той же аппаратуре. Спланируйте эксперимент с такой целью. Определите управляемые факторы и способы управления ими. Поможет ли вам знание различий между этими двумя системами (т. е. различий в стратегии планирования, стратегии распределения памяти) лучше спланировать эксперимент? Если да, то предположите, что вы знаете эти различия, и задайте их до описания эксперимента.

6.14. В некой установке с пакетной обработкой операционная система, используемая в настоящее время, выполняет только программы, полностью загруженные в основную память. Таким образом, если размер программы превышает максимально отведенный для нее размер памяти, программист должен разделить ее на сегменты и управлять выделенным пространством с помощью явных обращений к операционной системе. Операционная система, реализующая виртуальную память посредством автоматического управления двухуровневой иерархией памяти, может быть установлена без каких бы то ни было модификаций существующего оборудования. Спланируйте эксперимент, направленный на сравнение производительностей этих двух операционных систем. В частности, выберите индекс (или индексы) производительности и критерии, которых вы будете придерживаться при проектировании модели рабочей нагрузки. Отметим, что оценка производительности систем с виртуальной памятью обсуждается подробно, но под другим углом зрения в гл. 7 и 8. Цель этой задачи — предложить вам поразмыслить над ней с точки зрения сравнения производительностей. Вам, быть может, захочется вернуться к этой задаче после проработки следующих глав. Чтение работы Сейра (1969) может оказаться побудительным.

6.15. Широко распространенной единицей скорости ЦП является так называемый MIPS (миллион операций в секунду). Разумно ли и в какой мере сравнивать производительности различных ЦП посредством сравнения их скоростей в единицах MIPS, сообщаемых поставщиками? Почему?

6.16. Симс (1974) определил комбинированный фактор нагрузки для систем CP/67 и MTS (см. пример 6.5 и табл. 6.15) следующим образом:

$$L_{\text{CP-MTS}} = 2n_1 + 8n_2 + 6n_3 + 6n_4 + 10n_5 + n_6.$$

Вычислите значения других факторов нагрузки [см. (6.22) и (6.23)] для трех рабочих нагрузок, перечисленных в табл. 3.6.2. Воспользуйтесь этими результатами, чтобы показать, что средние времена ответа этих систем (которые приблизительно пропорциональны соответствующим факторам нагрузки) являются, по всей вероятности, многозначными функциями переменной $L_{\text{CP-MTS}}$.

6.17. Какая из двух кривых на рис. 6.8 представлялась бы вам более важной, если бы в 1973 г. вас попросили предсказать ежегодную внутриуниверситетскую рабочую нагрузку Дартмутской системы разделения времени на следующие 2 финансовых года?

6.18. Определите необходимую мощность ЦП в установке, описанной в примере 6.6, если устанавливаемая система должна работать без задержек, причем

a) ожидаемая ежегодная рабочая нагрузка будет создана коллективом пользователей только через 3 года работы новой системы;

Таблица 3.6.2

Рабочие нагрузки, соответствующие одному значению комбинированного фактора нагрузки ($L_{\text{CP-MTS}} = 100$) для эксперимента, описанного в примере 6.5

Номер сценария (см. табл. 6.15)	Число терминалов, работающих по каждому сценарию		
	Рабочая нагрузка X	Рабочая нагрузка Y	Рабочая нагрузка Z
1	4	2	3
2	1	5	4
3	—	6	—
4	4	—	9
5	5	1	—
6	10	10	8
Всего терминалов	24	24	24

b) пиковая ежедневная нагрузка, предположительно превышающая вдвое среднюю дневную нагрузку, будет достигнута также через 3 года.

6.19. Предположим, что система, мощность ЦП которой в четыре раза больше, чем в существующей системе, установлена в результате исследования, описанного в примере 6.6. Как вы должны были бы изменить предсказание относительно времени, когда новая система начнет работать с задержками, если данные, собранные за первый год работы, показали, что скорости роста r_i были занижены, а на самом деле они на 50% выше?

7
Оценка производительности в задачах усовершенствования

7.1. ЗАДАЧИ УСОВЕРШЕНСТВОВАНИЯ

Одной из наиболее важных областей применения методов оценки производительности является область усовершенствования. Предполагается, что объект исследования по усовершенствованию, например вычислительная система, работает в момент исследования в своей производственной среде, но уровень производительности не отвечает запросам администратора или пользователя. И даже если эти запросы удовлетворены, всегда удобно перевести систему на более высокий уровень производительности, так как при этом возрастет ее ценность для пользователя. Часто повышение производительности системы приносит и финансовые выгоды. Например, та же рабочая нагрузка может быть обработана за меньшее время, что позволяет сократить оперативный персонал; лучшая производительность привлекает больше покупателей; приобретение большей, более дорогой системы можно отложить (или совсем отказаться от ее покупки) благодаря возросшей способности существующей системы справляться с увеличенной рабочей нагрузкой; добавление какого-то аппаратного компонента к конфигурации может оказаться необязательным в результате изменений, предложенных в ходе исследования по усовершенствованию. Такое исследование может обнаружить, что некоторый аппаратный компонент (например, канал ввода-вывода) является излишним, так что его удаление не понижает существенно производительность системы. В этом случае мы получаем *снижение цены* вместо *повышения производительности*. Мы уже видели, что часто повышение производительности приводит к снижению цены. Более того, характеристики и методы работ по снижению цены весьма похожи на характеристики и методы исследований по повышению производительности. Таким образом, мы в этой главе будем подразумевать оба типа исследований, несмотря на то что чаще будем рассматривать исследования, главная цель которых состоит в повышении производительности.

Для того чтобы понять, имеет ли смысл проводить исследования, следует сравнить финансовые выгоды и преимущества от повышения производительности со стоимостью исследования. К сожалению, такое сравнение можно сделать лишь апостериори, когда результаты этого сравнения уже бесполезны. До исследования мы можем только приблизительно оценить (и должны это сделать) ожидаемые выгоды и затраты, чтобы определить, следует ли его предпринимать. Несмотря на то что в большинстве практических случаев возможны только очень гру-

бые оценки, всегда разумнее рассмотреть этот вопрос, чем начинать вслепую проводить исследование, которое может обернуться растратой ресурсов.

Можно ли определить *оптимальную производительность* вычислительной системы? И если да, то насколько трудно ее определить и достигнуть? В принципе оптимальная производительность, которую может обеспечить определенная система при обработке заданной рабочей нагрузки, может быть определена, если четко указаны параметры системы, которые могут меняться, и пределы их изменения. Например, мы увидим в разд. 7.3.2, что для мультипрограммной системы с виртуальной памятью обычно существует значение степени мультипрограммирования, максимизирующее среднюю пропускную способность системы для заданной рабочей нагрузки. Таким образом, если интересующий нас индекс производительности — средняя пропускная способность, а единственный параметр, который можно менять — степень мультипрограммирования, то определенно существует оптимальная производительность, которую может показать система. Однако практические ситуации редко бывают столь просты. Когда начинается работа по усовершенствованию, параметры системы, которые следует рассматривать в качестве кандидатов на изменение, едва ли могут быть перечислены. Кроме того, их границы трудно установить. В определенных случаях стоимость изменения, например расширения памяти, зависит от размера изменения и границы могут диктоваться максимальным доступным количеством имеющихся денежных средств и ценами остальных возможных сопутствующих изменений. Некоторым ограничениям нелегко придать количественную форму. Эти трудности часто делают практические ситуации совсем непохожими на точно поставленные математические задачи оптимизации. Даже если оптимум можно определить, его крайне трудно найти. Главными причинами неоптимальной производительности вычислительных систем являются высокая стоимость процедур оптимизации и относительно малые дополнительные преимущества по сравнению с аналогичными системами при увеличении производительности выше некоторого предела. Работа по усовершенствованию действительно будет успешной, если даст существенное увеличение производительности при относительно небольших затраченных усилиях. Если производительность не слишком далека от оптимальной, обычно ее можно увеличить лишь ненамного, и эти улучшения, как правило, обходятся довольно дорого. Таким образом, термин *усовершенствование* отражает существо дела намного лучше, чем термин *оптимизация*, и по этой причине он будет использоваться в этой книге вместо последнего.

Мы уже сказали выше, что обычные операционные условия в реальной вычислительной системе не являются оптимальными. Почему системы не проектируются так, чтобы обеспечивать оптимальную производительность при данной цене? Почему работы по усовершенствованию необходимы в столь многих случаях и так часто? По-видимому, существуют три главные причины этой ситуации: непривлекательность

проектирования по индивидуальным заказам с промышленной и коммерческой точек зрения; сложность вычислительных систем, для полного овладения которыми недостаточно наших знаний; изменчивость операционной среды системы. Производительность системы зависит от типа и величины рабочей нагрузки, которую она должна обрабатывать, и от процедур управления ею. Так как каждая установка имеет свою рабочую нагрузку и свои процедуры, то оптимальную для данной установки систему можно было бы в принципе спроектировать и построить на голом месте, но цена ее была бы недопустимой. Даже если бы эту проблему удалось преодолеть, сомнительно, что оптимальная система могла бы быть сконструирована, так как у нас нет точного понимания сложных взаимосвязей между индексами производительности и параметрами системы и рабочей нагрузки. Более того, даже если бы наших знаний было достаточно, динамическая природа среды большинства систем потребовала бы еще периодической настройки и сделала бы производительность системы неоптимальной для большой доли времени существования системы.

Вычислительные системы обычно конструируются объединением ранее разработанных компонентов оборудования и координацией их работы с помощью легко приспосабливаемого программного обеспечения (см. гл. 8). Относительная гибкость и совместимость компонентов оборудования и программного обеспечения нацелена на то, чтобы можно было получать разнообразные конфигурации посредством комбинирования различных специализированных модулей из одного резерва. Основное предположение состоит в том, что эти конфигурации можно приспособить к постоянно меняющимся требованиям различных установок. Следовательно, мы можем сказать, что работы по усовершенствованию не только желательны и часто очень выгодны, но также являются существенной частью процесса проектирования.

7.2. МЕТОДЫ УСОВЕРШЕНСТВОВАНИЯ

Тот же самый недостаток знаний, который так затрудняет проектирование вычислительной системы с оптимальной производительностью, обычно не позволяет нам определить за один шаг, можно ли улучшить производительность, насколько и что для этого следует сделать. Процедура усовершенствования неизбежно оказывается итеративной и в действительности содержит несколько циклов на различных уровнях. Работы, выполняемые в главном цикле, можно разбить на два основных этапа, для которых мы используем медицинские термины *диагностика* и *терапия*. Этим этапам часто, но не всегда, предшествует появление *симптомов* (рис. 7.1).

Исследование по усовершенствованию может быть инициировано администратором установки даже при отсутствии каких бы то ни было симптомов. В этих случаях начальная цель состоит в том, чтобы определить, есть ли возможность улучшения и оправдывает ли поиски ожидаемая величина улучшения. Чаще, однако, необходимость такого исследования следует из наблюдений, наводящих на мысль, что система не показывает такую производительность, как могла бы, или что производительность стала недостаточной. Эти симптомы могут поступать из таких источников, как суточная или месячная статистика, наблюдения операторов, жалобы пользователей и данные, собранные измерителями непрерывного наблюдения. Оперативное сообщение о симптомах проблем производительности является фактически вместе с учетом

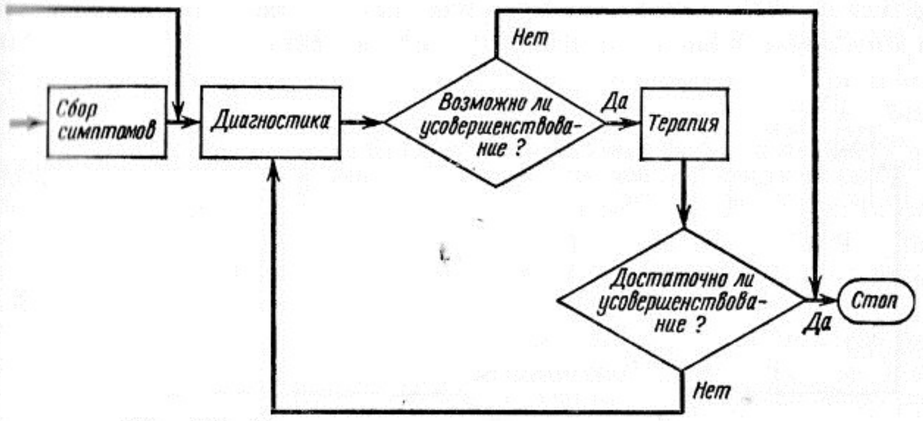


Рис. 7.1. Блок-схема процедуры усовершенствования.

главной целью непрерывного наблюдения. Учетная программа — простейший и всегда присутствующий в системе измеритель для непрерывного наблюдения. Данные, полученные этой программой, а также другими возможными дополнительными измерителями, должны выполнять функцию «тревоги». Типичными измерениями, отвечающими этим требованиям, являются измерения первичных индексов производительности и коэффициентов использования каждого компонента оборудования. Другие полезные переменные для непрерывного наблюдения можно найти среди параметров рабочей нагрузки. Однако область применения и разрешающая способность измерителей непрерывного наблюдения даже больше ограничены искажениями, чем у других измерителей, поскольку снижение производительности, которое они могут вызвать, является постоянным.

С другой стороны, как и в медицинской практике, диагноз в основном основывается на симптомах (если они есть). Если бы непрерывное наблюдение обеспечивало больше данных, чем это, строго говоря, необходимо, чтобы привлечь внимание администратора установки, то легче было бы понимать, в чем может состоять проблема. Однако с глобальной точки зрения, вероятно, удобнее отложить сбор дальнейших экспериментальных доказательств до этапа диагностики, во время которого этот сбор можно ограничить информацией, необходимой для проверки определенных, точно сформулированных гипотез. В этом контексте поставить диагноз системе значит определить, содержится ли в ней так называемый *дефект производительности*, где он локализован и как его можно устранить. Типичная диагностическая процедура будет состоять из

формулировки гипотезы, выполнения предварительного анализа ее разумности и ее проверки. Если результаты анализа на разумность или последующей проверки окажутся отрицательными, следует сформулировать другую гипотезу и повторять эту процедуру до тех пор, пока не будут получены положительные результаты или не будут исчерпаны возможности формулирования гипотез. В последнем случае мы должны прийти к заключению, что не можем улучшить систему и, следовательно, либо должны научиться обходиться ее существующим уровнем производительности, либо нам следует заменить систему на более

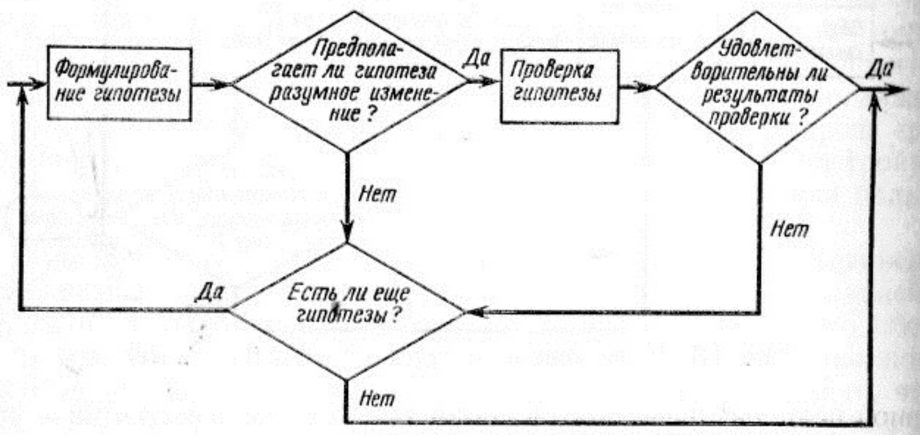


Рис. 7.2. Блок-схема этапа диагностики.

мощную. Не нужно пренебрегать теми гипотезами, из которых следует, что улучшение может быть достигнуто уменьшением величины или изменением природы рабочей нагрузки. Итеративная процедура, которую мы только что описали, навеяна работой Белла и др. (1972). Эта процедура показана на рис. 7.2. Ее следует представлять вставленной внутрь процедуры усовершенствования, изображенной на рис. 7.1, которая, в свою очередь, также является итеративной.

Сделаем несколько замечаний по блок-схеме рис. 7.2. Во-первых, Белл и др. (1972) отмечают, что формулированию гипотезы должна предшествовать попытка получения информации. Нельзя рассчитывать на то, что исследователи сразу же сформулируют полезные гипотезы, если они не обладают основательным знанием системы, если они не знают организации данной установки, ее рабочей нагрузки и процедур управления, если им не известны цели административного управления, а также симптомы или их эквиваленты. Сбор этой информации явно не упоминается на рис. 7.1 и 7.2, поскольку в некоторых случаях аналитик может уже хорошо знать данную установку. Например, существуют установки, персонал которых включает одного или нескольких специалистов, ответственных за оценку производительности. Однако утверждение, что чем лучше известна установка, тем вероятнее успех работы, необходимо подчеркнуть, несмотря на его очевидную справед-

ливость, так как аналитики иногда излишне полагаются на свою интуицию.

Во-вторых, должны быть сформулированы гипотезы о причинах более низкой производительности, чем ожидалось. Симптомами, если они есть, являются результаты. На этапе диагностики пытаются обнаружить причины. Иногда причины предполагаются косвенно. Формулируемая гипотеза состоит, скажем, не в том, что основная память слишком мала для данной конфигурации и рабочей нагрузки, а в том, что увеличение размера основной памяти дает наибольший вклад в увеличение производительности. Легко понять, что нет существенной концептуальной разницы между гипотезой о причинах и гипотезой о терапевтических мерах, которые могут оказаться эффективными при устранении явно не сформулированных причин. Как мы увидим, если гипотеза о причинах оказалась верна, терапия, которую следует применить, почти всегда очевидна, кроме всего прочего, из-за того, что проверка гипотезы часто состоит в прогнозировании результатов предлагаемой ею терапии.

Следует заметить, что формулирование гипотез — задача трудная и довольно тонкая. Как и в случае медицинского диагноза, это — искусство, и основой успеха здесь обычно является прочный фундамент знаний, понимания и опыта, а также интуиция. Основное — это знание зависимости индексов производительности от рабочей нагрузки и параметров системы, т. е. от факторов, влияющих на производительность. Не существует справочников, содержащих списки симптомов и соответствующие гипотезы, хотя имеются весьма полезные описания попыток построения списка примеров такого рода [см., например, Белл и др. (1972), Кокрум и Крокетт (1971)]. Белл и др. (1972) на основании своего опыта выделили пять общих методов, которые могут помочь формулировать корректные гипотезы. Эти методы перечислены и кратко прокомментированы в табл. 7.1. Мы проиллюстрируем применение некоторых из них при обсуждении примеров, приведенных далее в этой главе.

Третье замечание по блок-схеме на рис. 7.2 должно быть сделано относительно шага, следующего за формулированием гипотезы. Это тест на разумность терапии, предлагаемой данной гипотезой. Даже перед проверкой справедливости гипотезы имеет смысл посмотреть, окажутся ли возможные средства эффективными по стоимости. В этот момент стоимость изменения может быть оценена с достаточной точностью. Стоимость остальной части исследования предсказать труднее, но, по всей вероятности, можно оценить порядок ее величины. Экономичность и выгоды с точки зрения производительности, которые может дать терапия, можно оценить лишь грубо. Однако такой тест полезен, так как он может избавить исследователя от хлопот и затрат, связанных с проверкой гипотезы, предлагающей экономически неоправданные изменения. Разумеется, гораздо более точные оценки преимуществ окажутся возможными после того, как будет проверена справедливость гипотезы. Таким образом, наше предварительное заключе-

ние о разумности предложенного изменения следует проверить повторно перед завершением этапа диагностики, т. е. когда мы интересуемся, удовлетворительны ли результаты проверки. Примером неразумной гипотезы является гипотеза, требующая привлечения еще

Таблица 7.1

Некоторые общие методы формулирования гипотез

Метод	Описание
Идентификация похожих ситуаций	Свой собственный или заимствованный опыт проведения работ по усовершенствованию используется для генерации гипотез в случаях, похожих на уже известные
Обнаружение выделяющихся значений	Выделяющиеся значения индексов производительности часто являются интересными симптомами и могут привести к правильным гипотезам. Их следует игнорировать только после тщательного объяснения
Обнаружение закономерностей	Отыскиваются интересные закономерности во времени (циклы, тенденции). Очень полезны графические методы, но масштабы времени и отображаемые величины должны быть тщательно отобраны
Обнаружение корреляций	Корреляции между параметрами рабочей нагрузки или системы и индексами производительности могут привести к правильным гипотезам. Однако существование корреляций не всегда обусловлено причинно-следственными отношениями
Обнаружение и анализ несоответствий	Проверяется существование очевидных взаимосвязей между данными, собранными из различных источников, между данными, связанными с заданиями, и данными, связанными с системой, между измеренными данными и ожидаемыми их значениями. Обнаруженные несоответствия с большой вероятностью приведут к гипотезам

Источник: Белл и др. (1972).

одного оператора для ускорения обслуживания лентопротяжных устройств, в то время как это уменьшит среднее время прохождения всего лишь на 5%.

Последнее замечание надо сделать относительно шага проверки гипотезы на рис. 7.2. Этот шаг обычно включает получение дополнительных необходимых данных посредством специального измерения. Собранные данные должны обеспечить аналитика основанием для принятия гипотезы или отказа от нее. В идеале нам хотелось бы уметь предсказывать, какова будет производительность системы, если ее изменить в соответствии с испытываемой гипотезой. Сравнение этой производительности с первоначальной показало бы нам, верна ли гипотеза (см. рис. 7.3). Анализ производительности измененной системы относительно просто выполняется, если использовать подходящую имитационную или аналитическую модель этой системы. Выполнение этого исследования посредством измерения потребовало бы

изменения системы в действительности. В некоторых случаях, когда эти изменения малы, относительно дешевы и не слишком разрушительны, это можно делать. Например, влияние определенных изменений операционной системы на производительность можно проверить с помощью искусственных моделей рабочей нагрузки. Затем, прежде чем

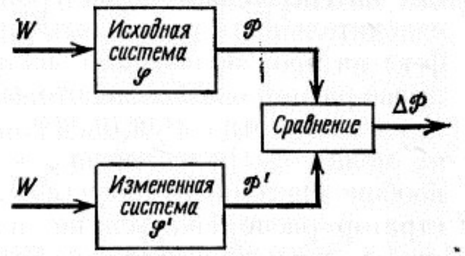


Рис. 7.3. Проверка гипотезы об усовершенствовании с помощью анализа производительности и сравнения. S и S' могут быть версиями существующей системы или их моделями, W может быть реальной рабочей нагрузкой или ее моделью.

внести эти изменения в производственную версию системы, их проверяют при производственной рабочей нагрузке в течение выделенных интервалов времени. В других случаях, например если бы гипотеза потребовала замены или добавления дорогого компонента оборудования, невозможно измерить эффекты этого изменения непосредственно перед принятием решения об изменении. Таким образом, справедливость гипотезы должна быть проверена либо методами моделирования, либо тщательным сбором и интерпретацией данных, либо обоими методами. Примеры этих методов обсуждаются в разд. 7.3. Как показано выше, результаты этого шага окажутся удовлетворительными, если выгоды, ожидаемые от предложенной терапии, явно превышают ее стоимость.

При диагностике системы всегда существует возможность, что ни одна из гипотез, сформулированных нами, не является одновременно разумной и справедливой (см. рис. 7.2). В этом случае, как показано на рис. 7.1, результат работы по усовершенствованию будет отрицательным и, если создавшаяся ситуация недопустима, придется рассмотреть другие альтернативы (новая более мощная система или дополнительная внутренняя или внешняя установка для поглощения части рабочей нагрузки). Если гипотеза оказалась справедливой, должен быть выполнен этап терапии. Это просто фактическое внесение в работающую систему изменений, которые, как ожидается, должны быть эффективны согласно сделанным прогнозам на этапе диагностики. Когда проверка «гипотезы-победителя» выполнена путем изменения операционной системы, этап терапии обычно проходит быстро и гладко. Если требуются изменения оборудования, этот этап оказывается гораздо дольше и сложнее. Может потребоваться работа по выбору новых компонентов, возможно, придется рассмотреть различные варианты закупок и может пройти существенный интервал времени, прежде чем выбранные компоненты будут поставлены и смонтированы.

Типы изменений, из которых может состоять улучшающая терапия, можно классифицировать различными способами. Во-первых, устано-

вим различие между *системными изменениями* и *программными изменениями*. Последняя категория включает изменения всех программ, выполняемых в системе, так чтобы улучшить их производительность (производительности программы будет дано определение и она будет обсуждаться в гл. 9). Для вычислительных систем часто оказывается справедливым утверждение Адама Смита о том, что отдельные частные интересы дают вклад в общее благо. Иначе говоря, улучшение производительности отдельных программ оказывает благоприятный эффект на производительность системы. Администратор установки имеет право распоряжаться некоторыми из программ, выполняемых системой. К их числу относятся операционная система (которая, однако, иногда не может быть изменена по условиям контракта), компиляторы и вообще системные программы. Программами пользователей администратор распоряжаться не может. Администраторы установок часто могут влиять на выполнение заметной части программ. Они могут также создать стимул для повышения производительности программы пользователя, сурово наказывая «плохие» программы. Таким образом, изменения программ всегда следует рассматривать среди возможных методов терапии, которые может использовать администратор для улучшения производительности системы. Специфические методы, которые можно использовать для повышения производительности программы, мы рассмотрим в гл. 9. В этой главе, в частности в разд. 7.4, мы сосредоточим внимание на изменениях в системе. Методы терапии в этой категории можно подразделить на *изменения конфигурации оборудования* и *изменения операционной системы*. Последние методы терапии будут называться *методами настройки*, а первые — *методами развития*.

Под настройкой многие понимают подгонку операционной системы под действительную рабочую нагрузку существующей системы. Мы обсуждали необходимость этой операции в конце разд. 7.1. Рабочие нагрузки проявляют тенденцию к изменению во времени, их объем обычно растет, а их природа медленно или резко меняется. Таким образом, систему необходимо перенастраивать несколько раз на протяжении ее существования. Когда обнаруживаются симптомы падения производительности, предпочтение отдают гипотезам, которые приводят к настройке, поскольку настройка дешевле развития. К сожалению, когда существующая конфигурация оборудования близка к насыщению, настройка практически бесполезна и становится необходимым развитие (или полная замена). Иногда, как отмечено в разд. 7.1, исследование может привести к исключению из конфигурации некоторого компонента оборудования. В этом случае улучшаются экономические характеристики системы, а не ее производительность.

Как уже отмечалось несколько раз в предыдущих главах, любые изменения, влияющие на производительность системы, вообще говоря, меняют поведение коллектива пользователей и, следовательно, рабочую нагрузку. Несмотря на то что в ходе работ по оценке обычно не принимают во внимание эту обратную связь (см. разд. 5.1), некоторые изменения операционных систем, направленные на изменение рабочей нагрузки, могут улучшить производительность. Большинство таких изменений не использует механизм обратной связи, упомянутый выше. Например, объем рабочей нагрузки можно уменьшить, не разрешая некоторым пользователям доступ к системе. Уменьшить объем нагрузки можно также посредством ручного или автоматического выбрасывания заданий, когда система перегружена. С помощью обратной связи

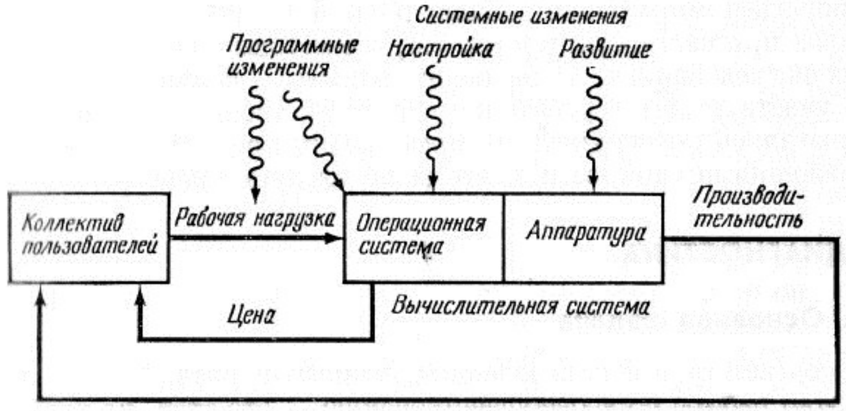


Рис. 7.4. Типы методов терапии усовершенствования.

«производительность/рабочая нагрузка» можно предотвращать использование системы теми пользователями, которые ответственны за низкую производительность системы, или побуждать их улучшать производительность своих программ. Аналогичный тип терапии, относящийся не к обратной связи «производительность/рабочая нагрузка», а скорее к обратной связи «стоимость/рабочая нагрузка», состоит в изменении политики назначения цен таким образом, чтобы лучше приспособить рабочую нагрузку к характеристикам системы. До некоторой степени этого можно было бы добиться, повышая плату за интенсивно используемые ресурсы и понижая ее для слабо нагруженных устройств. Результаты обсуждения этого этапа терапии графически представлены на рис. 7.4.

Как показано на рис. 7.1, за проведением терапии должна следовать оценка ее воздействия на производительность системы. Произошло ли на самом деле предсказанное улучшение? Из-за приближений, сделанных при проверке гипотез, реальное улучшение может оказаться совсем не таким, как ожидавшееся. Если удовлетворительный уровень производительности не был достигнут, необходимо вернуться к этапу диагностики, или собрать симптомы измененной системы, или, может быть, исследовать более радикальные альтернативы. Сравнение производительностей часто является существенной частью и этого шага. Характеристики сравнения в контексте усовершенствования, как показано на рис. 7.3, полностью отличаются от характеристик, с которыми мы встречались в задачах выбора. Системы, которые предстоит сравнивать, являются на самом деле версиями одной и той же системы, от-

личающимися часто только в очень немногих (но существенных) деталях. Такая ситуация в значительной степени определяет выбор используемых методов оценки, а также проблемы описания рабочей нагрузки, которые предстоит решить. Эти вопросы обсуждаются в разд. 7.5 и 7.6.

Какова роль оценки производительности в исследованиях по усовершенствованию? В отличие от роли, которую она играет в процедурах выбора, здесь это крайне важная роль. Как описано в последующих разделах и, в частности, в разд. 7.5, все методы оценки используются в этих исследованиях. Производительность — главное для исследователя, участвующего в работе по усовершенствованию. Однако, как мы уже пытались подчеркнуть на протяжении предыдущего обсуждения, соображениями стоимости никогда не следует пренебрегать.

7.3. ДИАГНОСТИКА

7.3.1. Основная задача

В соответствии с определением, данным в разд. 7.2, диагностический этап работы по усовершенствованию — это этап, на котором формулируются и проверяются гипотезы о причинах неудовлетворительной производительности или по крайней мере о предпочтительной терапии для улучшения. В обосновании надобности этапа диагностики неявно присутствует очень важное соображение, что различные изменения вычислительной системы обычно оказывают решающее влияние на ее производительность. При этом возможны такие изменения, которые не оказывают никакого влияния в одних ситуациях и дают значительный эффект в других. Их влияние на индексы производительности зависит от конфигурации и от рабочей нагрузки. Они могут оказывать сильное воздействие на некоторые индексы и не оказывают никакого на остальные. Таким образом, изменения, которые, вероятно, дадут заметный положительный результат, предстоит определять последовательно. Кроме того, предстоит выбрать среди этих изменений те, которые обладают наилучшими характеристиками «стоимость-производительность».

Эти эффекты обусловлены нелинейностями, характеризующими поведение вычислительных систем. Точнее, пусть $\mathcal{S}$ — система, P — индекс производительности, а $x_1, x_2, \ldots, x_n$ — n параметров, которые мы считаем изменяемыми. Величины x_i включают параметры рабочей нагрузки наряду с параметрами системы. Предположим, что увеличение P соответствует повышению производительности. Функция $P(x_1, x_2, \ldots, x_n)$ обычно характеризуется ярко выраженными нелинейностями. Эти нелинейности часто принимают форму *узких мест*. Неформально, мы говорим, что $\mathcal{S}$ имеет узкое место относительно P в определенной области пространства своих изменяемых параметров, если в этой области значение P существенно возрастает при изменении только одного или немногих параметров. Даже большие изменения остальных параметров не способны заметно увеличить P, пока они не настолько велики, чтобы вывести систему за пределы этой области. В нашем неформальном определении узкого места мы, разумеется, предполагали, что возможно увеличение P внутри этой области. Таким образом, мы исключили из рассмотрения области, окружающие глобальные или локальные максимумы P. Если система обладает узким местом в области, в которой она работает, единственный путь положительного воздействия на P состоит в том, чтобы сначала правильно диагностировать узкое место, т. е. определить, какие именно из изменяемых параметров лимитируют P в этой области. Тогда можно воспользоваться терапией, состоящей в изменении должных параметров в правильном направлении и на наиболее подходящую величину.

Чтобы понять, почему в вычислительных системах могут возникнуть узкие места, нужно рассмотреть внутреннюю структуру и поведение этих систем. Их описание по типу «черного ящика», представленное функцией $P(x_1, x_2, \ldots, x_n)$, недостаточно для этой цели. Вычислительную систему можно рассматривать как набор ресурсов, запрашиваемых проходящими через нее заданиями. Каждое задание для завершения своего выполнения требует определенное количество определенных ресурсов оборудования и программного обеспечения. Ресурсы ограничены, и только очень небольшое количество заданий может использовать каждый данный ресурс в каждый данный момент времени. Например, только одно задание может выполняться в ЦП в каждый данный момент времени. Порядок, в котором задание запрашивает необходимые ресурсы, в большинстве случаев фиксирован. Таким образом, задание, запрашивающее некоторый ресурс, который нельзя предоставить немедленно, должно в общем случае ждать, пока этот ресурс не будет освобожден всеми заданиями, обладающими более высокими приоритетами на использование данного ресурса. Если задание одно в системе, то ресурсы, которые оно требует, всегда свободны и очередей нет. Поэтому в однопрограммных системах не бывает узких мест. Узкие места возникают в мультипрограммных системах, когда запросы на ресурсы от одновременно выполняемых заданий не сбалансированы. В этой ситуации существуют ресурсы, запросы на которые поступают чаще, чем позволяет скорость их обслуживания системой. Таким образом, за этими ресурсами выстраиваются относительно длинные очереди, а в результате другие ресурсы остаются недоиспользованными. Разумеется, перегрузку можно уменьшить изменением некоторых выбранных параметров системы или параметров рабочей нагрузки, или и тех и других.

Мультипрограммирование было введено с целью увеличения операционной мощности вычислительных систем. Разрешение полного перекрытия операций различных компонентов системы является потенциально эффективным методом ускорения выполнения заданной рабочей нагрузки. Однако, для того чтобы потенциальные преимущества мультипрограммирования стали реальностью, должна быть достигнута ситуация равномерной загрузки этих компонентов. Не должно быть пре-

пятствий для равномерного потока заданий от одного компонента к другому — иными словами, не должно быть узких мест. Заметим, что идеальное условие, с точки зрения разумного использования мультипрограммирования, состояло бы в том, чтобы все независимые компоненты были использованы на 100%. Это означало бы, что система производит максимальное количество работы. Однако, как отмечалось в разд. 1.4, максимизация определенного индекса производительности (скажем, средней пропускной способности) может вывести за допустимые границы другой индекс (скажем, среднее время ответа). Мы вернемся к этой проблеме позже в данном разделе.

Подход к усовершенствованию, предполагавшийся в приведенном выше обсуждении, мог бы быть назван *последовательным устранением*

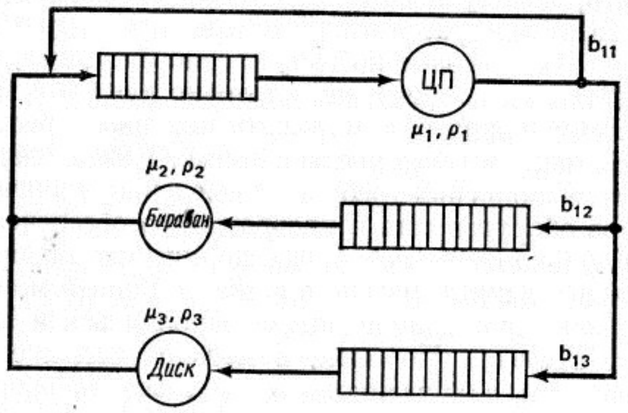


Рис. 7.5. Модель центрального обслуживания для системы, исследуемой в примерах 7.1, 7.2, 7.3, 7.10, 7.11, 8.1, 8.2, 8.4, 8.5 и 8.6.

узких мест. Терапия, которая действует на один или несколько параметров, вообще говоря, не всегда приводит к балансировке системы. Могут обнаружиться новые узкие места, которые ранее были скрыты только что устраненным. Если ищется возможность дальнейшего улучшения, эти узкие места должны быть диагностированы и «излечены». Кроме того, иногда применение терапии может перевести систему в другую область пространства параметров, где производительность ограничивается другим узким местом. Читатель, владеющий основами математического программирования, может легко обнаружить концептуальное сходство между этим итеративным подходом и методом градиента в применении к функции $P(x_1, x_2, \ldots, x_n)$. Однако ситуация осложняется тем, что часто набор изменяемых параметров включает нечисловые, например представляющие стратегию управления ресурсами.

Концепция узкого места и влияние узких мест на производительность системы будут теперь проиллюстрированы простым аналитическим примером.

Пример 7.1. Ориентированную на пакетную обработку версию ПРОП-системы, включающей ЦП, барабан и диск с подвижными головками, можно приблизительно моделировать сетью с центральным обслуживающим прибором, с очередями, показанной на рис. 7.5 (см. разд. 4.3.5) Система является мультипрограммной для постоянного числа N заданий, которые находятся во время обработки в основной памяти. Используется стратегия постоянного распределения памяти. Каждое задание загружается в раздел постоянного размера ($32K$ слов) в начале шага задания и ос-

Таблица 7.2

Параметры системы и рабочей нагрузки для модели с рис. 7.5

Параметр	Описание	Значение
	(a) Параметры системы	
$r_{ЦП}$	Средняя скорость выполнения команд ЦП	10^6 команд/с
M	Размер основной памяти для заданий пользователей	$160K$ слов
m	Размер раздела памяти	$32K$ слов
$r_{бр}$	Скорость вращения барабана	3000 об/мин
$t_{бр}$	Время пересылки слова на барабан или из барабана	3 мкс
$r_{де}$	Скорость вращения диска	1500 об/мин
$t_{поиск}$	Среднее время поиска на диске	70 мс
$t_{де}$	Время обмена словом с диском	15 мкс
	(b) Параметры рабочей нагрузки	
I	Среднее число команд, которые должны быть выполнены за один шаг задания	240 000 команд
$n_{ЦП}$	Среднее число интервалов времени ЦП на шаг задания	20
$l_{бр}$	Среднее число слов, передаваемых за одно обращение к барабану	2000 слов
$n_{бр}$	Среднее число обращений к барабану за шаг задания	15
$l_{де}$	Среднее число слов, передаваемых за одно обращение к диску	2000 слов
$n_{де}$	Среднее число обращений к диску за шаг задания	4

вобождает этот раздел только в конце шага. Для простоты мы будем рассматривать различные шаги некоторого задания как различные задания. Таким образом, не будет разницы между заданиями и шагами заданий. Как и в разд. 4.3.5, предположим, что задания статистически идентичны, а их запросы на время обслуживания распределены по экспоненциальному закону. Мы также предполагаем, что нет постоянно пустых разделов. В качестве индекса производительности в нашем анализе будет рассматриваться средняя пропускная способность, задаваемая формулой (4.103):

$$\overline{T} = b_{11}\mu_1\rho_1. \tag{7.1}$$

В табл. 7.3 значения параметров модели получены из значений соответствующих параметров системы и рабочей нагрузки, перечисленных в табл. 7.2. Средняя пропускная способность может быть вычислена по формуле (7.1), если известен коэффициент использования ЦП ρ_1. Значение ρ_1 для модели с центральным обслуживающим прибором (как на рис. 7.5) можно получить методом, рассмотренным в разд. 4.3.5. Применяя этот метод и уравнения (4.108) и (7.1), получим $\rho_1 = 0.475$, $\rho_2 = 0.475$, $\rho_3 = 0.95$ и $\overline{T} = 1.98$ заданий/с. Если бы ЦП был использован полностью, средняя пропускная способность была бы $\overline{T}_{max} = b_{11}\mu_1 = 4.16$ заданий/с.

Теперь мы хотим исследовать возможность увеличения $\bar{T}$ посредством изменения одного или более параметров системы или рабочей нагрузки, которые появляются в нашей модели. Изменяемые параметры перечислены в табл. 7.4. Заметим, что они вклю-

Таблица 7.3

Параметры модели, полученные из параметров, приведенных в табл. 7.2

Параметр	Уравнение	Значение
N	$N=M/m$	5
μ_1	$\mu_1=\bar{v}_{\text{ЦП}}n_{\text{ЦП}}/\bar{L}$	83.33 c^{-1}
μ_2	$\mu_2=1/[(60/2R_{\text{бр}})+t_{\text{сбр}}\bar{r}_{\text{бр}}]$ *	62.5 c^{-1}
μ_3	$\mu_3=1/[(60/2R_{\text{дс}})+\bar{t}_{\text{поиск}}+t_{\text{сдс}}\bar{r}_{\text{дс}}]$ *	8.33 c^{-1} **
b_{11}	$b_{11}=1/\bar{n}_{\text{ЦП}}$	0.05
b_{12}	$b_{12}=\bar{n}_{\text{бр}}/\bar{n}_{\text{ЦП}}$	0.75
b_{13}	$b_{13}=\bar{n}_{\text{дс}}/\bar{n}_{\text{ЦП}}$	0.2

Эта таблица используется в нескольких примерах в гл. 7 и 8. Значения параметров в третьем столбце различаются от примера к примеру, а иногда и внутри одного примера. Эти отличия всегда указаны в тексте.

* $t_{\text{сбр}}$, $t_{\text{сдс}}$ и $\bar{t}_{\text{поиск}}$ должны быть выражены в секундах.

** В примерах 7.1 и 7.2 (частично), а также в примерах 7.3, 7.10 и 7.11 это значение заменено на 16.66 c^{-1}.

чают не все указанные в табл. 7.2. Используя метод, описанный в разд. 4.3.5 для вычисления ρ_1, мы можем вычислить значение $\bar{T}$, даваемое моделью любой измененной версии нашей системы. Результаты приведены в виде графика на рис. 7.6. Из этого

Таблица 7.4

Изменяемые параметры для примера 7.1

Изменение	Изменяемые параметры	Изменяемые параметры модели
(a) *Развитие*		
Размер основной памяти	M	N
Скорость ЦП	$\bar{v}_{\text{ЦП}}$	μ_1
Скорость барабана	$R_{\text{бр}}$, $t_{\text{сбр}}$	μ_2
Скорость диска	$R_{\text{дс}}$, $t_{\text{сдс}}$, $\bar{t}_{\text{поиск}}$	μ_3
(b) *Настройка*		
Переупорядочение файлов на диске	$\bar{t}_{\text{поиск}}$	μ_3
Перемещение файлов на другое устройство	$\bar{n}_{\text{бр}}$, $\bar{n}_{\text{дс}}$	b_{12}, b_{13} *

* Заметим, что $b_{12}+b_{13}=1-b_{11}=\text{const}$.

рисунка очевидно, что система обладает узким местом. Увеличение интенсивности обслуживания ЦП (μ_1), барабана (μ_2) или размера памяти (N) не оказывает значительного влияния на $\bar{T}$. Единственными параметрами среди перечисленных в табл. 7.4, изменение которых может вызвать существенное увеличение производительности, яв-

ляются μ_3 и b_{13} (или соответственно b_{12}). Таким образом, мы приходим к выводу, что диск создает узкое место и что единственными путями повышения $\bar{T}$ являются либо уменьшение нагрузки диска (путем непосредственного уменьшения $\bar{t}_{\text{поиск}}$ или b_{13}), либо повышение скорости диска (путем увеличения $R_{\text{дс}}$ и, как следствие, уменьшения $t_{\text{сдс}}$ или уменьшения $\bar{t}_{\text{поиск}}$), либо и то и другое. Первый метод терапии относится к настроечному типу, а последний требует развития системы, т. е. замены диска на более быстрый. Графики на рис. 7.6 показывают, кроме того, что когда мы изменяем один или более параметров с целью ликвидации узкого места, скорость увеличения производительности резко уменьшается за определенной точкой. Достигнув этой точки, мы можем сказать, что узкое место уже практически устранено, а производительность

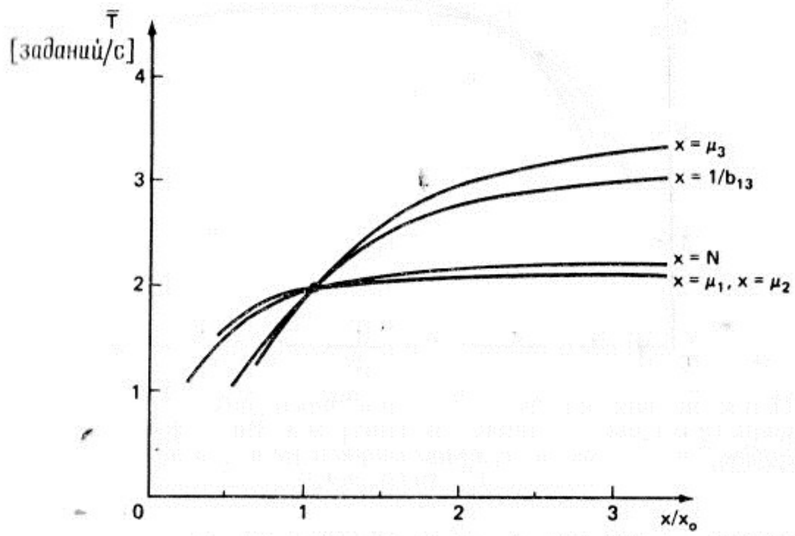


Рис. 7.6. Изменения средней пропускной способности модели рис. 7.5, соответствующие изменениям различных системных параметров x. Вид параметра x указан около каждой кривой; x_0 — значение x в неизменной системе (см. табл. 7.3).

более не лимитируется данными параметрами. Если предположить, что обнаруженное узкое место «излечивается» увеличением скорости обслуживания диска, то, как видно из рис. 7.6, эта точка соответствует скорости обслуживания диска, приблизительно вдвое большей, чем у существующей системы. Если мы изменим систему в соответствии с нашим диагнозом, ее модель будет иметь те же значения параметров, что и в табл. 7.3, кроме параметра μ_3, который теперь будет равен 16.66 c^{-1}. Коэффициенты использования трех обслуживающих приборов в этой модели все равны 0.714. Средняя пропускная способность оказывается $\bar{T}=2.97$ задания в секунду. На рис. 7.7 приведены графики $\bar{T}$ как функции изменяемых параметров для усовершенствуемой системы. Заметим, что, если в новой системе скорость ЦП или барабана уменьшена в 2 раза, получается то же самое значение $\bar{T}$, как в первоначальной системе. Заметим также, что графики рис. 7.7 не показывают очевидного наличия узкого места. Мы вернемся к этой проблеме в разд. 7.3.2 после того, как дадим аналитическое определение узкого места. Другое интересное замечание, которое необходимо сделать относительно рис. 7.7, относится к форме кривой $1/b_{13}$. Если разрешена только настроечная терапия, существует такое размещение файлов, которое максимизирует $\bar{T}$. Это не удивительно, так как сумма b_{12} и b_{13} должна быть постоянной, и, если b_{13} убывает, то b_{12} должен расти. Таким образом, эффект уменьшения b_{13} в некотором смысле эквивалентен эффекту увеличения скорости диска при одновременном уменьшении скорости барабана. В некоторой точке уменьшение производительности, обусловленное барабаном (который может стать узким местом), становится больше,

чем увеличение, обусловленное сокращением нагрузки на диск. Что действительно может вызвать удивление, так это несовпадение положения оптимума $\bar{T}$ с точкой равномерного использования всех обслуживающих приборов. Это явление будет исследовано в разд. 7.3.2.

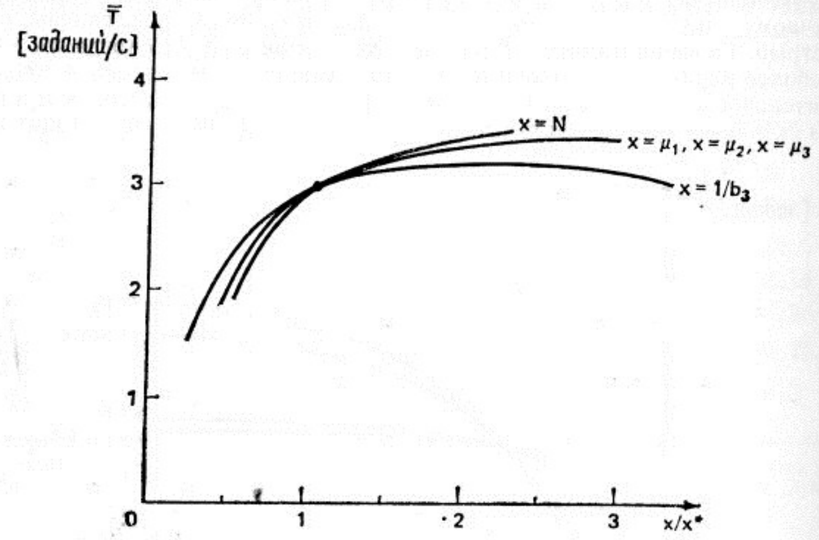


Рис. 7.7. Изменения средней пропускной способности модели рис. 7.5, соответствующие изменениям различных системных параметров x. Вид параметра x указан около каждой кривой; x^* — значение x, приведенное в табл. 7.3, кроме значения для μ_3: $\mu_3 = 16.66$ с$^{-1}$.

В примере 7.1 мы рассмотрели влияние изменений так называемых изменяемых параметров системы на ее среднюю пропускную способность. Наш анализ был основан на модели центрального обслуживающего прибора для простой системы пакетной обработки. Нам теперь хотелось бы изучить влияние тех же самых изменений на другие интересующие нас индексы производительности, в частности типа индексов реактивности, таких как среднее время прохождения $\bar{t}$. Зависимости коэффициентов использования от изменения параметров и их строгая взаимосвязь со средней пропускной способностью будут исследованы в разд. 7.3.2. Теперь мы проанализируем поведение $\bar{t}$ как функции изменяемых параметров в версии ПРОП-системы, которую мы рассматривали в примере 7.1:

Пример 7.2. Моделируя нашу систему вновь, как на рис. 7.5, мы можем определить $\bar{t}$ как среднее реальное время между моментом, когда задание впервые поступает в очередь к ЦП, и моментом, когда оно покидает систему по возвратному пути вокруг ЦП (см. рис. 7.5). Как показано в разд. 4.3.5 [уравнение (4.104)], мы получим

$$\bar{t} = \frac{N}{\bar{T}} . \tag{7.2}$$

Таким образом, когда N, т. е. размер памяти M, постоянно, $\bar{t}$ обратно пропорционально $\bar{T}$. Каждому увеличению $\bar{T}$ будет соответствовать улучшение реактивности системы. Например, значения параметров в табл. 7.2 дают $\bar{t} = 2.52$ с. Если скорость

обслуживания у диска увеличится вдвое, среднее время прохождения станет $\bar{t} = 1.68$ с. Ресурсы, создающие узкие места по отношению к $\bar{T}$, играют ту же самую роль по отношению к $\bar{t}$.

Однако, если изменить размер памяти, приведенные выше утверждения перестанут быть верными. На рис. 7.8 представлены графики зависимости $\bar{T}$ и $\bar{t}$ от N для пер-

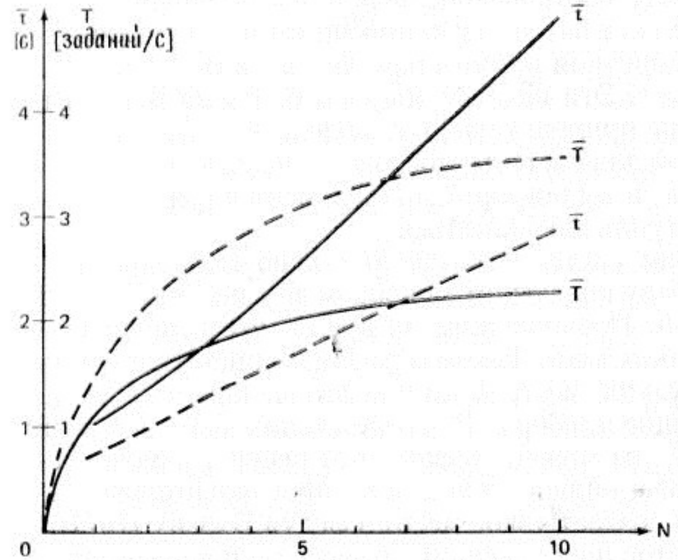


Рис. 7.8. Средняя пропускная способность $\bar{T}$ и среднее время прохождения $\bar{t}$ как функции числа заданий N (размера памяти) для модели рис. 7.5. и табл. 7.3. Пунктирные кривые относятся к случаю $\mu_3 = 16.66$ с$^{-1}$.

вначальной и измененной систем. В то время как абсолютные и относительные эффекты изменения скорости обслуживания у диска растут с ростом N, улучшение $\bar{T}$ сопровождается ухудшением $\bar{t}$ и наоборот. Из (7.2) видно, что произведение $\bar{t}\bar{T}$ растет линейно с ростом N. $\bar{T}$ не может расти линейно с ростом N из-за узких мест, созданных другими ресурсами. Таким образом, как только $\bar{T}$ перестает расти, $\bar{t}$ должно увеличиваться.

Из этого примера можно заключить, что продуктивность и реактивность часто могут расти одновременно, хотя и не всегда. Если принята терапия усовершенствования, приводящая к более тяжелой нагрузке на все ресурсы, так что все очереди могут стать длиннее, продуктивность растет за счет реактивности.

В этом разделе мы ввели понятие узкого места, которое имеет первостепенную важность на этапе диагностики процедуры усовершенствования, объяснили, почему возникают узкие места в вычислительных системах и показали на простой аналитической модели, как они проявляются. Более глубокое изучение этой модели, включая формальное определение узкого места и основания для обсуждения формулировки гипотезы, содержится в следующем разделе. В разд. 7.3.3 описано применение этих и других более эмпирических методов формулирования гипотез к измерениям или к результатам моделирования. Наконец, в разд. 7.3.4 приведена статистическая диагностическая процедура.

7.3.2. Аналитическое исследование узких мест

Обсуждение узких мест в разд. 7.3.1 и изучение результатов, полученных на модели из рис. 7.5, которое мы проделали в примере 7.1, выдвинули ряд вопросов. Как можно аналитически определить узкие места? Каковы соотношения между коэффициентами использования компонентов системы и существующими в ней узкими местами? Максимальна ли средняя пропускная способность и в каком смысле, когда нет узких мест? Эти важные вопросы будут затронуты в первой части этого раздела применительно к модели с центральным обслуживающим прибором для системы, подобной уже рассмотренной в примере 7.1. Во второй части мы повторим анализ, используя ту же модель для системы с виртуальной памятью.

Рассмотрим систему, которую можно моделировать сетью с центральным обслуживающим прибором и s центрами обслуживания (на рис. 7.5 $s=3$). Предположим на время, что основная память системы не создает узких мест. Таким образом, единственными возможными источниками узких мест, явно представленными в модели, являются обслуживающие приборы. В соответствии с наблюдениями, сделанными в разд. 7.3.1, мы можем сказать, что ресурс, изменение которого оказывает наиболее сильное влияние на производительность, и есть именно тот, который создает узкое место. Следуя Бузену (1971), мы определим в качестве источника узкого места ресурс, представленный обслуживающим прибором Σ_i, таким, что

$$\frac{\partial \overline{T}}{\partial \mu_i} > \frac{\partial \overline{T}}{\partial \mu_j} \quad (j=1,\ 2,\ \ldots,\ s;\ j \neq i). \tag{7.3}$$

Заметим, что из этого определения не следует, что узкое место будет полностью или частично устранено при действительном увеличении интенсивности обслуживания прибора Σ_i. Настройку (например, подходящие изменения b_{1i} в нашей модели) можно также использовать для изменения ситуации.

Из приведенного выше определения непосредственно следует, что если

$$\frac{\partial \overline{T}}{\partial \mu_i} = \frac{\partial \overline{T}}{\partial \mu_j} \tag{7.4}$$

для всех i и j, то в системе нет узких мест. В этом случае система называется *сбалансированной*. В сбалансированной системе (небольшое) увеличение интенсивности обслуживания одним из компонентов увеличит $\overline{T}$ на одну и ту же величину независимо от того, какой именно компонент используется. Несмотря на различия горизонтальных масштабов кривых на рис. 7.6, очевидно, что наша система со значениями параметров из табл. 7.2 не сбалансирована, а диск — источник узкого места.

Каково соотношение между сбалансированной системой и системой, в которой все ресурсы равномерно используются? А с системой, в которой в точке равновесия длины всех очередей равны? А с системой, имеющей максимальную пропускную способность?

Условия равномерного использования

$$\rho_i = \rho_j \quad (i,\ j=1,\ 2,\ \ldots,\ s), \tag{7.5}$$

если вспомнить уравнения (4.108), примут вид

$$b_{1i} = \frac{\mu_i}{\mu_1}, \qquad \frac{b_{1i}}{b_{1j}} = \frac{\mu_i}{\mu_j} \quad (i,\ j=2,\ \ldots,\ s). \tag{7.6}$$

На рис. 7.7 представлена как раз такая ситуация в рассматриваемой нами системе. Кроме того, можно показать (см. Бузен (1971)), что если условия (7.6) и, следовательно, (7.5) выполнены, все средние длины очередей равны. Однако такая ситуация, вообще говоря, не является сбалансированной, в которой мы имели бы, согласно (7.4) и (4.103),

$$\frac{\partial \rho_1}{\partial \mu_i} = \frac{\partial \rho_1}{\partial \mu_j} = \frac{\rho_1}{\mu_1} + \frac{\partial \rho_1}{\partial \mu_1} \quad (i,\ j=2,\ \ldots,\ s). \tag{7.7}$$

Рис. 7.7, например, показывает, что если мы учтем различие горизонтальных масштабов и начнем с точки равных коэффициентов использования ($x/x^*=1$), увеличение $\overline{T}$, обусловленное данным увеличением интенсивности обслуживания диском, оказывается больше, чем увеличения вследствие повышения интенсивности обслуживания как ЦП, так и барабаном. Это также подтверждается тем, что, начиная с этой же точки, $\overline{T}$ растет с уменьшением b_{13} (и увеличением b_{12}). Таким образом, точка баланса могла бы быть достигнута при дальнейшем увеличении μ_3 до тех пор, пока частные производные T по μ_3 окажутся равными остальным частным производным в соответствии с условием (7.4). Следует, однако, отметить, что изменение только μ_3 может оказаться недостаточным, чтобы удовлетворить (7.4). Необходимо также подбирать интенсивности обслуживания остальных приборов.

Сбалансированная система не использует всего своего потенциала производительности. Среднюю пропускную способность сбалансированной системы обычно можно повысить настройкой, даже если эта операция нарушает баланс. Таким образом, если задано сочетание компонентов оборудования и определенной рабочей нагрузки, максимальное значение средней пропускной способности обычно соответствует несбалансированной ситуации. Бузен (1971) нашел, что в точке этого максимума сохраняются для всех $i,\ j=2,\ \ldots,\ s$ следующие соотношения:

$$\frac{b_{1i}}{b_{1j}} = \frac{1+\overline{N}_i-(\overline{N}_i/\rho_i)}{1+\overline{N}_j-(\overline{N}_j/\rho_j)}, \tag{7.8}$$

где $\overline{N}_i$ означает среднюю длину очереди к центру обслуживания Σ_i (включая задание, возможно, уже обслуженное), и

$$\frac{\rho_i}{\rho_j} = \frac{\partial \overline{T}/\partial \mu_i}{\partial \overline{T}/\partial \mu_j}. \tag{7.9}$$

Уравнение (7.8) показывает, что если Σ_i и Σ_j имеют одинаковые коэффициенты использования в максимуме $\overline{T}$, т. е. если $\rho_i=\rho_j$ и $\overline{N}_i=\overline{N}_j$, то мы должны получить $b_{1i}=b_{1j}$. Если в максимуме $\overline{T}$ мы имеем $b_{1i}\neq b_{1j}$, то Σ_i и Σ_j не могут иметь в максимуме одинаковых коэффициентов использования. Таким образом по уравнению (7.9) там должно быть узкое место. Точнее, если $\mu_i>\mu_j$, то мы имеем $b_{1i}>b_{1j}$, когда $\overline{T}$ максимально. Так как эти два неравенства подразумевают $\rho_i>\rho_j$, то из уравнения (7.9) следует, что более быстрый обслуживающий прибор Σ_i создает в этой точке (относительное) узкое место. Единственный случай, когда нет узких мест в максимуме,— случай равных коэффициентов использования всех обслуживающих приборов [см. (7.9)]. Однако, согласно (7.8), это может случиться, только если все b равны между собой и, согласно (7.6), если все μ равны. Заметим, что утверждения, сделанные в этом абзаце, неприменимы к центральному обслуживающему прибору. Например, читатель может легко проверить, используя (4.108), что в точке максимума $\overline{T}$ на рис. 7.7 $\rho_2>\rho_1$. Иначе говоря, узкое место обусловлено барабаном, а не центральным процессором, хотя центральный процессор быстрее.

Влияние размера памяти на производительность до сих пор игнорировалось. Определение узкого места, данное в (7.3), можно было бы обобщить, чтобы включить также ресурс памяти. Однако сравнение $\partial\overline{T}/\partial N$ с остальными частными производными имеет мало смысла, так как они имеют различную размерность. Эта проблема обсуждается с более практической точки зрения в разд. 7.4.3. Здесь мы изучим поведение $\overline{T}$ и коэффициентов использования в зависимости от N.

Из (4.108) мы имеем

$$\frac{\rho_i}{\rho_1}=\frac{\mu_1 b_{1i}}{\mu_i}. \tag{7.10}$$

Поскольку правая часть (7.10) не зависит от N, это же должно быть справедливо для отношения ρ_i/ρ_1. Таким образом, *относительные коэффициенты использования* обслуживающих приборов $\Sigma_2, \ldots, \Sigma_s$ остаются постоянными при изменении N. При росте N средние длины очередей будут расти, а обслуживающие приборы будут использоваться больше. Согласно (7.10), коэффициенты использования будут расти пропорционально, так что их отношения останутся неизменными. Однако ни одно значение ρ не станет больше 1. Таким образом, поскольку максимальное ρ_i $(1\leqslant i\leqslant s)$ приближается к 1 асимптотически, остальные ρ будут стремиться к меньшим значениям. Соотношения (7.10) сохраняются также между этими асимптотическими значениями. На рис. 7.9 приведены графики зависимостей ρ от N для модели, обсужденной в примере 7.1. Так как ρ_3 (коэффициент использования диска, создающего узкое место) стремится к 1, то ρ_1 и ρ_2 стремятся только к 0.5. Не удивительно, что изменение N не воздействует на узкое место, созданное обслуживающими приборами в системе. Единственным случаем, когда все коэффициенты использования стремятся к 1 с ростом N, является случай равенства коэффициентов использования всех обслуживающих приборов.

Согласно (4.103), $\overline{T}$ пропорционально ρ_1. Так как $\rho_1(N)$ — монотонно возрастающая функция, то $\overline{T}(N)$ — также монотонно возрастает и имеет ту же форму. Как показано на рис. 7.6, 7.7 и 7.8, $\overline{T}$ стремится

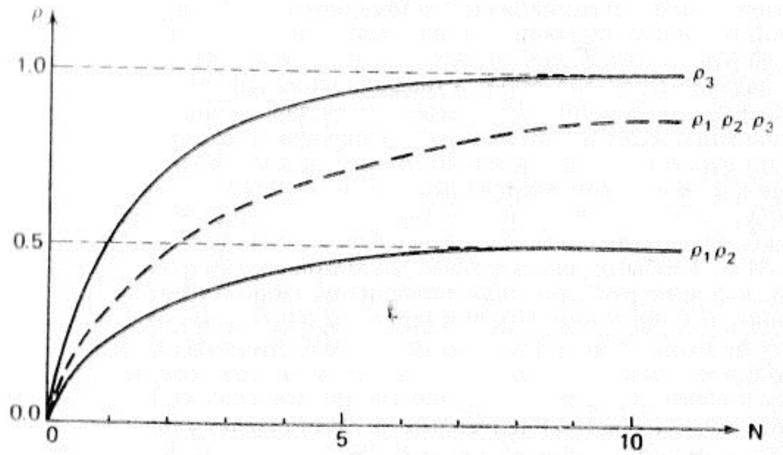


Рис. 7.9. Коэффициенты использования обслуживающих приборов в модели рис. 7.5 и табл. 7.3 как функции числа заданий в системе. Пунктирная кривая относится к случаю $\mu_3=16.66$ с$^{-1}$.

асимптотически к $b_{11}\mu_1\rho_1(\infty)$, совпадающему с b_{11}/μ_1, абсолютным максимумом пропускной способности системы, только если $\rho_1(\infty)=1$ (т. е. если ЦП — самый интенсивно используемый ресурс). Пунктирная линия $\overline{T}$ на рис. 7.8 стремится к максимальной емкости, тогда как сплошная линия $\overline{T}$ стремится к половине этого значения. Читатель может заинтересоваться, действительно ли асимптота пунктирной линии, соответствующая условию равенства коэффициентов использования, представляет абсолютный максимум емкости системы. Ранее в этом разделе мы уже видели, что максимальное значение $\overline{T}$ не обязательно достигается, когда все ρ равны. Однако Бузен (1971) показал, что с ростом N расстояние между точкой максимума $\overline{T}$ и точкой равенства ρ уменьшается. При $N\to\infty$ эти точки стремятся к одному пределу. Таким образом, с ростом памяти (и степени мультипрограммирования) система автоматически стремится к настроенному состоянию.

Некоторые из уже полученных результатов можно применить к изучению узких мест в системах с виртуальной памятью. Чтобы упростить рассуждения, мы проанализируем на примере специально модифицированную модель такой системы с центральным обслуживающим прибором.

Пример 7.3. Система, которую мы будем изучать, это мультипрограммная версия ПРОП-системы пакетной обработки с виртуальной памятью. Компоненты системы, явно присутствующие в модели, те же, что и в примере 7.1: один ЦП, один барабан (используемый в качестве страничного устройства) и диск (для системных фай-

лов и файлов пользователей). Основная память содержит только части выполняемых заданий и управляется автоматически. Пересылки между памятью и страничным барабаном не запрашиваются явно заданиями, а происходят тогда, когда необходимая информация не обнаружена в памяти. Мы предполагаем, что виртуальная память организуется средствами страничной обработки (см. пример 2.7). Память разделена на *страничные области*, каждая из которых содержит одну *страницу* информации, принадлежащую некоторому заданию. Обмен информацией между памятью и барабаном осуществляется постранично. Когда страница, содержащая виртуальный адрес, на который сослалась выполняемая программа, отсутствует в памяти, генерируется страничный отказ, выполнение задания прерывается, ЦП переключается на готовое к работе задание (если оно есть), а операционная система выдает запрос барабану на отсутствующую страницу. Если вся память заполнена, отсутствующая страница должна заместить одну из страниц, находящихся в памяти. *Алгоритм замещения* выберет эту страницу. В данном примере мы предположим, что система использует *стратегию фиксированного распределения памяти*. Как и в примере 7.1, память разделена на N областей, каждая из которых отведена определенному заданию. Так как в нашей модели задания статистически идентичны, разумно сделать все их одного размера $m=M/N$. Чтобы изменить степень мультипрограммирования N, мы можем изменить число и размер областей при постоянном M или изменить M при постоянном m, или изменить одновременно M и m так, чтобы изменилось отношение M/m.

Основное отличие этой системы от системы, рассмотренной в примере 7.1, состоит в том, что изменение размера памяти, отведенного заданию, изменяет динамику страничного обмена («подкачки»), причем в некоторых случаях весьма существенно. Таким образом, если мы меняем степень мультипрограммирования N, сохраняя M постоянным, это изменит в нашей модели b_{12}, а следовательно, и другие b. Так как продолжительность интервалов времени ЦП меняется при изменении частоты страничных отказов, то μ_1 также изменится (см. уравнение для μ_1 в табл. 7.3, которое содержит $\bar{n}_{цп}$). Заметим, что все эти изменения происходят постольку, поскольку изменяется частота запросов к барабану, т. е. частота страничных отказов в программах. Следовательно, как мы увидим далее, они могут быть обусловлены изменениями не только параметра m, но также и стратегии управления памятью или порядка обращений к памяти у программ.

Чтобы изучить влияние степени мультипрограммирования на $\bar{T}$ в нашей модели, мы вначале предположим, что N изменяется посредством изменения M, а m остается постоянным. При этих условиях легко определить поведение $\bar{T}$ в зависимости от N, соответствующее любой данной интенсивности обращений к барабану для каждого задания. Поскольку m и интенсивность обращений к барабану для каждого задания постоянны (в нашей модели) при изменениях N, модель совпадает с изученной в примере 7.1, а $\bar{T}$ растет монотонно по N, стремясь к горизонтальной асимптоте, как показано на рис. 7.6, 7.7 и 7.8. При значениях параметров, перечисленных в табл. 7.3, за исключением μ_3, которое мы полагаем равным 16.66 $с^{-1}$, $\bar{T}$ будет следовать кривой (a) на рис. 7.10. Эта кривая совпадает с пунктирной кривой для $\bar{T}$ на рис. 7.8. Если мы теперь изменим либо поведение программ, либо алгоритм замещения, либо и то и другое, так что среднее число обращений к барабану на одно задание окажется равным $\bar{n}_{бр}=195$, то получим $\bar{n}_{цп}=1+\bar{n}_{бр}+\bar{n}_{дс}=200$. Согласно уравнениям в табл. 7.3, получим $\mu_1=833.3\ с^{-1}$, $b_{11}=0.005$, $b_{12}=0.975$ и $b_{13}=0.02$. При этих значениях параметров модели величина $\bar{T}$ следует кривой (b) на рис. 7.10. Из-за значительного возрастания страничных обменов барабан оказывается узким местом, что резко уменьшает производительность. Заметим, что размер страницы предполагался равным $1K$ слов и что каждый отказ потребует пересылки $2K$ слов, т. е. одна страница на барабан и одна с барабана (см. табл. 7.2).

Теперь пусть общий размер памяти M остается постоянным, а изменять N будем, меняя m, т. е. размер области памяти, отведенной каждому заданию. Известно, что когда m уменьшается, программы порождают большее количество страничных отказов. Следуя Деннингу и Грехему (1975), рассмотрим сначала следующую очень грубую модель поведения программы: если одной из наших программ выделено более 50 страниц, среднее число страничных отказов у нее равно $\bar{n}_{бр}=15$; если ей отводит

менее 50 страниц, то $\bar{n}_{бр}$ скачком возрастает до 195. Если бы было так, то средняя пропускная способность следовала бы кривой (a) на рис. 7.10 для $N=M/m\leqslant 3$, а для $N\geqslant 4$ она следовала бы кривой (b). Таким образом, график $\bar{T}$ оказался бы составленным из отрезков кривых (a) и (b). Средняя пропускная способность достигала бы максимума при степени мультипрограммирования, равной 3, и падала бы до очень

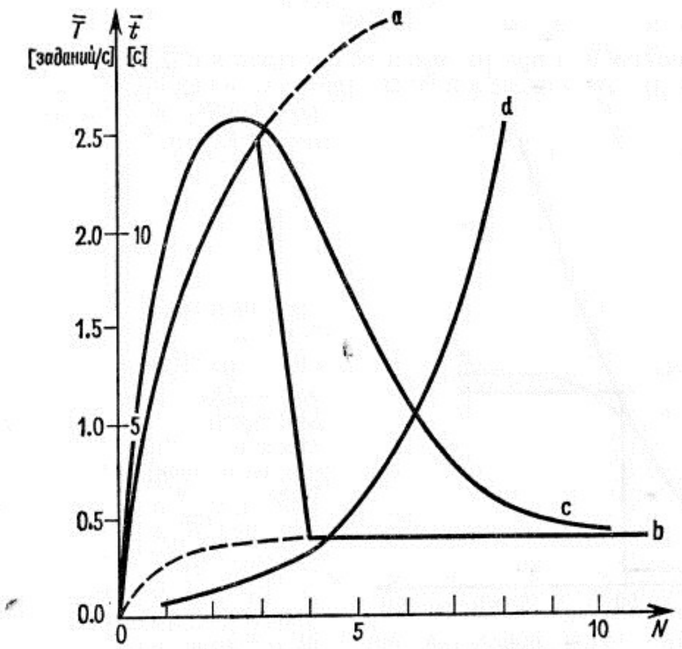


Рис. 7.10. Средняя пропускная способность $\bar{T}$ и среднее время прохождения $\bar{t}$ как функции степени мультипрограммирования N для модели системы с виртуальной памятью, проанализированной в примере 7.3: (a)— кривая $\bar{T}$ для $m=32K$ слов и $\bar{n}_{бр}=15$; (b) — кривая $\bar{T}$ для $m=32K$ слов и $\bar{n}_{бр}=195$; (c) — кривая $\bar{T}$ для $M=160K$ слов, $a=6.25$ страниц$^{-k}$, $v_{цп}=10^6$ команд/с, $k=2$; (d) — кривая $\bar{t}$ для того же случая, что и (c). Отметим, что $\mu_3=16.66с^{-1}$.

низких значений при $N\geqslant 4$. Это явление, называемое *толкотней* (thrashing), характерно для систем с виртуальной памятью и является одной из главных проблем, которые приходится решать конструкторам и настройщикам таких систем [см. Деннинг (1968)]. Падение кривой $\bar{T}(N)$ обычно не бывает столь резким, как это предполагалось в нашей примитивной модели. Однако толкотню можно часто рассматривать как катастрофическую потерю производительности. Ее причину легко вывести из предыдущего обсуждения. Вследствие увеличения степени мультипрограммирования выше определенного предела, размер области памяти, отведенной каждому заданию, становится недостаточным для размещения существенной информации, необходимой для выполнения задания. Следовательно, интенсивности страничных отказов для выполняемых заданий резко возрастают, что делает устройство страничной памяти узким местом, резко снижающим производительность.

Лучшее приближение к реальной кривой $\bar{T}$ для этой системы можно получить, немного усовершенствовав нашу модель поведения программы. Следуя Брандвайну и др. (1973), мы можем воспользоваться *функцией времени жизни*, предложенной Биледи и Кюнером (1969) в качестве аналитической аппроксимации эмпирических наблюдений, сделанных на ряде реальных программ. При фиксированной стратегии размещения и заданном алгоритме замещения было найдено, что средний интервал вре-

мени ЦП между последовательными страничными отказами $\bar{e}$ связан с m, как показывает кривая (a) на рис. 7.11. Кривая была названа *кривой времени жизни* программы Биледи и Кюнером (1969), которые предположили, что ее форма относительно нечувствительна к алгоритму замещения. Функция времени жизни

$$\bar{e} = \frac{a}{v_{\text{цп}}} m^k, \tag{7.11}$$

где a и k зависят от свойств программы, изображена на рис.7.11, кривая (b),для $a=6.25$ страницы $^{-k}$, $v_{\text{ЦП}}=10^6$ команд/с и $k=2$. Для нескольких программ экспериментально найдено, что показатель степени k лежит между 1.5 и 2.5. Так как

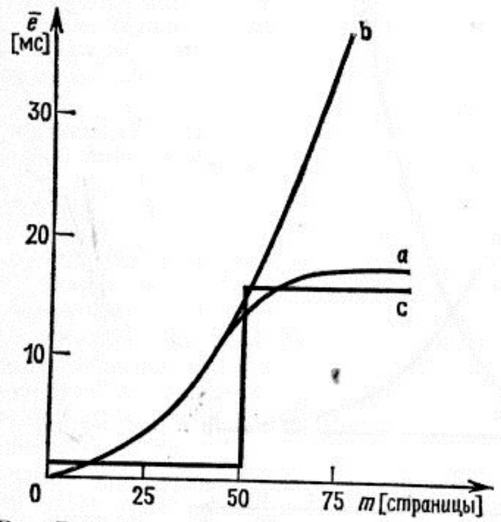


Рис. 7.11. Простые модели поведения программы: (a) — кривая времени жизни, (b)— функция времени жизни при $a=6.25$ страниц$^{-k}$, $v_{\text{цп}}=10^6$ команд/с, $k=2$; (c) — грубая модель, использованная в примере 7.3 [см. кривые (a) и (b) на рис. 7.10].

$$\bar{e} = \frac{\bar{L}}{\bar{v}_{\text{цп}}\ \bar{n}_{\text{бр}}}, \tag{7.12}$$

где использованы обозначения из табл. 7.2, наша предыдущая модель поведения программы представляется ступенчатой функцией на плоскости $(m, \bar{e})$ [см. кривую (c) на рис. 7.11]. Уравнения табл. 7.3 и (7.11) показывают, что при возрастании N величина m уменьшается, $\bar{n}_{\text{бр}}$ и $\bar{n}_{\text{цп}}$ возрастают; следовательно, b_{11} и b_{13} убывают, в то время как b_{12} возрастает. Так как произведение $\mu_1 b_{11}$ не зависит от N, то $\bar{T}$ пропорционально ρ_1 согласно уравнению (7.1). Метод, описанный в разд. 4.3.5 для вычисления ρ_1, можно применить и здесь, но для каждого N значения μ_1, b_{11}, b_{12} и b_{13} отличаются и должны быть пересчитаны с помощью уравнений табл. 7.3 и при

$$\bar{n}_{\text{бр}} = \frac{\bar{L}}{\bar{v}_{\text{цп}}\ \bar{e}}, \quad \bar{n}_{\text{ЦП}} = 1 + n_{\text{бр}} + \bar{n}_{\text{дс}}, \tag{7.13}$$

и $\bar{e}$ из (7.11).

Результаты, основанные на использовании функции жизни рис. 7.11 [кривая (b)], изображены на рис. 7.10 [кривые (c) и (d)]. Кривая (c) является более гладкой, чем кривая, полученная комбинированием кривых (a) и (b), но все же имеет максимум при $N=3$ и относительно быстрый спад $\bar{T}$ для $N \geqslant 4$. При $N=8$ $\bar{T}$ уменьшается до одной четверти максимального значения. Кривая (d) представляет собой график среднего времени прохождения, вычисленного по (7.2). Читатель может заметить, как резко оно возрастает после точки максимума $\bar{T}$. Коэффициенты использования в данной модели изменяются с N уже не так, как изображено на рис. 7.9. Графики ρ_1 и ρ_3 имеют ту же форму, что и график $\bar{T}$ [кривая (c) на рис. 7.10]. Отметим, что значение ρ_2 меньше ρ_1 для $N \leqslant 3$, но растет гораздо быстрее, чем ρ_1. Оно становится больше ρ_1 при $N=4$ и стремится к 1, в то время как ρ_1 и ρ_3 убывают. Таким образом, относительная степень использования устройства страничной памяти зависит от N. В действительности ρ_2/ρ_1 очень быстро возрастает с ростом N.

Можно предотвратить попадание системы с виртуальной памятью в режим толкотни посредством управления ее степенью мультипрограммирования. Заметим, что поддержание постоянного значения N нельзя назвать хорошим решением, так как

характеристики рабочей нагрузки меняются со временем, а следовательно, и кривая толкотни [кривая (c) на рис. 7.10] меняется со временем. Даже в нашей весьма упрощенной модели эта кривая зависит от значений a и k. Обсуждение этой проблемы управления будет проведено в разд. 7.4.2. Очевидно, возникновение толкотни можно также сделать более редким путем расширения полосы пропускания канала и устройства страничного обмена, т. е. путем увеличения μ_2. Результаты изменения μ_2 будут теми же, что и в примере 7.1. Возрастание $\bar{T}$, вызванное увеличением μ_2, оказывается значительно большим, если страничный барабан является узким местом. Когда μ_2 убывает, кривая опускается, особенно в области толкотни, а максимум смещается в сторону меньших значений N. Падение $\bar{T}$ из-за толкотни становится резче, тем самым подтверждая наблюдения Деннинга (1968 b), что чувствительность производительности к перегрузке памяти увеличивается с увеличением суммарного вре-

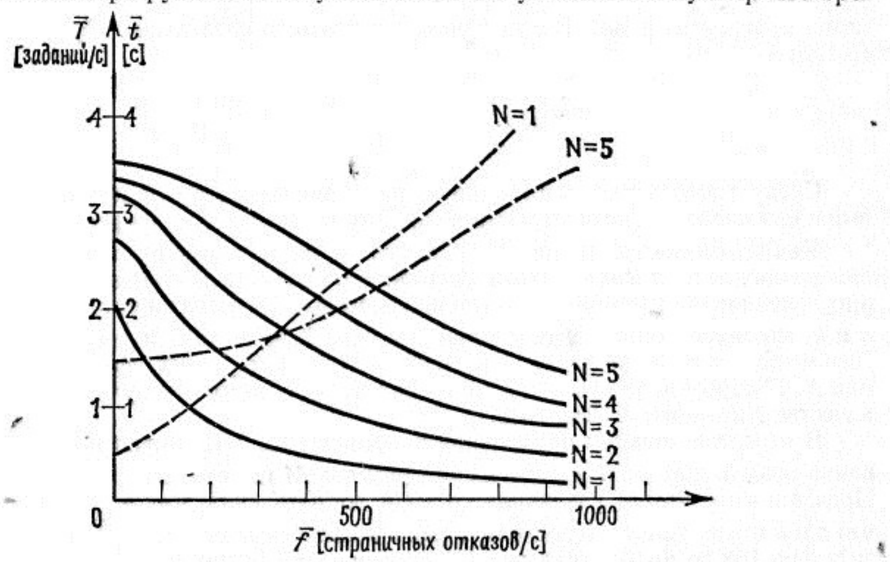


Рис. 7.12. Средняя пропускная способность $\bar{T}$ (сплошные линии) и среднее время прохождения $\bar{t}$ (пунктирные линии) как функции суммарной средней интенсивности страничных отказов $\bar{f}$ при различных значениях степени мультипрограммирования N для модели системы с виртуальной памятью, изученной в примере 7.3 ($\mu_3=16.66$ с$^{-1}$).

мени встречных пересылок между основной памятью и устройством внешней страничной памяти. Если μ_2 увеличивается, кривая всюду поднимается, особенно если возникает толкотня. Ее максимум соответствует большему значению N. Результаты увеличения μ_3 в некотором смысле противоположны. Относительное увеличение $\bar{T}$ оказывается много больше для малых N, особенно если при этом диск оказывается узким местом, чем для больших N, где узким местом оказывается страничный барабан. Эта же причина создаст тенденцию к сдвигу максимума кривой толкотни влево. Изменение $\bar{v}_{\text{цп}}$ влияет на $\bar{e}$ (см. 7.12) и на μ_1 (см. табл. 7.3), но не на b. Таким образом, результаты изменения $\bar{v}_{\text{цп}}$ легко можно предсказать. Например, увеличение $\bar{v}_{\text{цп}}$ всегда вызывает увеличение $\bar{T}$, но величина прироста зависит от того, является ли ЦП узким местом или нет. При увеличении N этот прирост будет уменьшаться, поскольку $\bar{T}$ будет все сильнее и сильнее ограничиваться страничным барабаном. Для малых значений N этот прирост, разумеется, будет меньше, если диск является узким местом, чем если им окажется ЦП. В любом случае увеличение $\bar{v}_{\text{цп}}$ будет приводить к уменьшению оптимальной степени мультипрограммирования. Ускорение одного из остальных компонентов способствует превращению страничного барабана в ресурс, лимитирующий производительность с ростом N.

Как влияет поведение программ на производительность системы с виртуальной памятью? А как влияет стратегия управления памятью? Эти вопросы обсуждаются в разд. 7.4.1 (см. пример 7.10) и 7.4.2. Предварительный ответ на оба вопроса можно дать с помощью нашей модели и с учетом того, что изменения поведения программ и стратегии управления памятью приводят к тем же результатам, что и изменения m, т. е. они изменяют количество страничных отказов в каждой программе и, следовательно, интенсивность ее обращений к барабану. Результаты изменения $\bar{n}_{бр}$ можно увидеть на рис. 7.10, сравнивая кривую (a) с кривой (b). Два более информативных и более полезных семейства кривых представлены на рис. 7.12. Они представляют поведение индексов производительности $\bar{T}$ и $\bar{t}$ при изменениях средней интенсивности страничных отказов $\bar{f}$. Значения параметров модели таковы, какими мы предполагали их на протяжении этого примера. Так как средняя интенсивность страничных отказов программы равна $1/\bar{e}$, то $\bar{f}$ можно вычислить следующим образом [см. (7.10) и (7.12)]:

$$\bar{f} = N\frac{1}{\bar{e}} = \frac{\bar{v}_{цп}\bar{n}_{бр}N}{\bar{L}} = \frac{v_{цп}N^{k+1}}{aM^k}. \qquad (7.14)$$

Следует заметить, что интенсивность страничных отказов программы определена выше как число запросов страниц в единицу времени ЦП, а не реального времени.

Как можно видеть на рис. 7.12, данное значение $\bar{f}$ дает различные значения $\bar{T}$ и $\bar{t}$ для различных степеней мультипрограммирования. Если мы знаем, каково влияние определенного изменения поведения программы или стратегии управления памятью на a и k, мы можем определить влияние на $\bar{f}$ (по 7.14) и на $\bar{T}$ и $\bar{t}$ по кривым рис. 7.12. Разумеется, этот метод можно использовать для изучения эффектов изменений M или N. Построение кривых (c) и (d) на рис. 7.10 по этому методу оставляем читателю в качестве упражнения (см. задачу 7.4).

В качестве примера применения графиков рис. 7.12 можно назвать прогнозирование воздействий на $\bar{T}$ и $\bar{t}$ изменений размера M памяти для заданий пользователя. Предположим, что M увеличили со $160K$ до $208K$ слов, что N остается равным 3 и что a и k имеют такие же значения, как и выше. Согласно (7.14), $\bar{f}$ убывает от приблизительно 168 до приблизительно 100 страничных отказов в секунду. Таким образом, из рис. 7.12 мы получаем, что $\bar{T}$, по-видимому, увеличится со значения, примерно равного 2.6, до приблизительно 2.9 задания в секунду, а $\bar{t}$ уменьшится от приблизительно 1.2 с до почти 1 с. Если при большей памяти мы хотим увеличить степень мультипрограммирования до 4, то (7.14) дает нам общую интенсивность 237 страничных отказов в секунду, а из рис. 7.12 мы получаем $\bar{T} \approx 2.8$ задания в секунду и $\bar{t} \approx 1.4$ с. ■

7.3.3. Обнаружение узких мест

Понятие узкого места было введено в разд. 7.3.1, а аналитически определено и изучено в разд. 7.3.2. В данном разделе мы зададимся вопросом, как можно обнаружить узкое место в работающей системе.

Чтобы обнаружить узкое место, мы должны сформулировать гипотезу о его существовании и обусловить и затем проверить справедливость этой гипотезы. Формулированию гипотезы весьма помогают симптомы. Каковы симптомы узкого места?

Аналитическое определение, которое было дано в разд. 7.3.2 [см. (7.3)], в общем случае не может помочь при обнаружении узких мест. В реальной системе часто оказывается трудно или невозможно оценить по результатам измерений частные производные индексов производительности по параметрам системы или рабочей нагрузки. Скорость процессора или размер памяти обычно невозможно увеличить специально для целей эксперимента такого типа. Иногда их можно уменьшить, но обычно только на дискретную сравнительно большую величину. Это, например, имеет место в случае для системы с несколькими блоками основной памяти или с несколькими дисками, один или несколько из которых можно исключить из системы, не нарушая ее работоспособности. Что касается параметров рабочей нагрузки, их можно менять контролируемым образом, только когда система управляется моделью искусственной рабочей нагрузки. Даже если в принципе можно использовать метод обнаружения, основанный на повторном использовании набора эталонов или синтетических заданий (см. задачу 7.2), стоимость многократного исполнения этой модели рабочей нагрузки часто делает ее непривлекательной. Эксперименты с управляемой рабочей нагрузкой, по-видимому, гораздо более удобны для проверки гипотезы. Заметим, что обсуждение, проведенное в этом абзаце, неприменимо к исследованиям на имитационной модели. Вычисление частных производных, когда доступна имитационная модель изучаемой системы, является принципиально простой задачей. Однако из соображений стоимости может вновь оказаться более привлекательным другое решение.

Достаточно хорошее решение, которое легко использовать, по крайней мере на первом шаге,— это решение, основанное на коэффициентах использования различных компонентов. Измерение этих факторов, как видно из гл. 2, в большинстве случаев относительно просто. В соответствии с выводами, которые мы получили в разд. 7.3.2, система с равномерно используемыми компонентами оборудования обычно не является ни сбалансированной, ни настроенной. Однако те три точки, в которых удовлетворено одно какое-то из этих условий, в общем случае находятся недалеко друг от друга. В частности, мы видели, что расстояние между точкой равного использования и точкой максимума $\bar{T}$ стремится к нулю с возрастанием степени мультипрограммирования. Гораздо важнее отметить, что большие различия между коэффициентами использования являются серьезным симптомом наличия узкого места, особенно если значение одного из этих коэффициентов близко к 1. Таким образом, хотя наблюдаемые значения коэффициентов использования не указывают прямо, является ли система сбалансированной или настроенной, они могут непосредственно указать исследователю на существование и причину возникновения явно узкого места. Примеры будут приведены ниже.

Другим полезным симптомом наличия узкого места является обнаружение того, что производительность системы заметно ниже ее возможного уровня. Знание этого не помогает установить причину узкого места. Однако возможность оценить потенциальную производительность, с которой можно было бы обрабатывать рабочую нагрузку дан-

ной установки, позволит администраторам легко обнаруживать проблемы производительности, требующие «лечения». Она даст им еще и мерило, с которым можно сравнивать достигнутые улучшения и оценивать дальнейшие усилия по усовершенствованию. Например, если производительность системы лишь на несколько процентов ниже предсказанного максимума, часто не имеет смысла начинать или продолжать работу по усовершенствованию.

Но как определить потенциальную производительность в данном контексте? Какие параметры считать изменяемыми? Рабочая нагрузка считается заданной и неизменной, хотя это предположение очень часто оказывается неверным. Предположим также, что наш главный индекс производительности — средняя пропускная способность. При очень грубых предположениях, на которых основано моделирование мультипрограммной системы сетью с центральным обслуживающим прибором с экспоненциально распределенными временами обслуживания и с очередями, из (7.1) и из табл. 7.3 можно получить следующее выражение для емкости системы:

$$\overline{T}_{max}=\frac{\bar{v}_{цп}}{\bar{L}}\rho_{1max}. \qquad (7.15)$$

Если приемлемы все изменения конфигурации, кроме влияющих на скорость обработки центральным процессором, мы получим $\rho_{1max}=1$ и $\overline{T}_{max}=\bar{v}_{цп}/\bar{L}=1/\bar{t}_c$, где $\bar{t}_c$ — средний запрос времени ЦП на задание, который легко может быть получен из учетных данных. Аналогично, (7.1), (4.108) и табл. 7.3 позволяют нам выразить $\overline{T}_{max}$ как функцию от μ_2, $\bar{n}_{бр}$ и ρ_{2max}:

$$\overline{T}_{max}=\frac{\mu_2}{\bar{n}_{бр}}\rho_{2max}. \qquad (7.16)$$

Если единственным неизменяемым параметром оказывается μ_3, то $\rho_{2max}=1$, а $\overline{T}_{max}=\mu_2/\bar{n}_{бр}$. Те же самые рассуждения, повторенные для интенсивности обслуживания диском, дают

$$\overline{T}_{max}=\frac{\mu_3}{\bar{n}_{дс}}\rho_{3max}, \qquad (7.17)$$

что для $\rho_{3max}=1$ сводится к $\overline{T}_{max}=\mu_3/\bar{n}_{дс}$. Для системы, изученной в примере 7.1, со значениями параметров из табл. 7.2 и 7.3 емкости, определенные по (7.15), (7.16) и (7.17), равны 4.166 заданиям в секунду для первых двух и 2.083 для третьего. Ясно, что наименьшее из этих чисел совпадает с емкостью системы, если единственной модификацией, принимаемой во внимание, является степень мультипрограммирования (см., например, рис. 7.8).

Если конфигурация оборудования должна оставаться фиксированной и разрешена только настроечная терапия, вычисление емкости оказывается гораздо сложнее. Мы обсудим эту проблему в конце разд. 7.4.1.

Минимальное значение среднего времени прохождения можно приблизительно вычислить по (7.2), как только получено $\overline{T}_{max}$. Заметим, что абсолютный минимум времени прохождения задания — время, которое задание проводит в системе, работающей в однопрограммном режиме. В примере 7.1 абсолютный минимум равен 0.96 с, а минимум при $N=5$ и единственном неизменяемом параметре μ_1 или μ_3 равен 1.2 с.

Гипотезы, обсуждавшиеся до сих пор, принадлежат к числу простейших самого элементарного типа. Как правило, диагностический этап не заканчивается обнаружением существования и локализацией узкого места. Обычно должны выдвигаться более тонкие гипотезы для идентификации причины узкого места, так чтобы можно было выбрать правильную терапию. Например, предположим мы обнаружили, что диск является узким местом. Это означает, что существует несоответствие между интенсивностью обслуживания диском и интенсивностью поступления запросов на диск. Однако из этого не следует, что скорость диска недостаточна и ее следует увеличить, это как раз один из возможных методов терапии, хотя на самом деле и дорогой. Другими методами терапии могут быть увеличение размера буфера, понижение степени мультипрограммирования, изменение или реорганизация содержимого диска, увеличение размера основной памяти или изменение алгоритмов планирования выполнения заданий или распределения диска. Эффективность этих методов зависит от истинной причины узкого места. Например, увеличение размера буфера уменьшает нагрузку на диск, только если существующий размер слишком мал для заданий, составляющих рабочую нагрузку. Легко представить себе ситуации, в которых такое увеличение было бы вредным. В соответствии с уравнениями в табл. 7.3, уменьшение $\bar{n}_{дс}$ могло бы с избытком компенсироваться уменьшением μ_3 из-за того, что $\bar{r}_{дс}$ увеличилось бы. Формулирование таких более тонких гипотез гораздо сильнее зависит от системы, чем для случая обнаружения узкого места. Общее и строгое рассмотрение этого крайне затруднено и, во всяком случае, очень тесно связано с соображениями терапии. Мы приведем некоторые примеры такого формулирования в разд. 7.4. Однако существуют некоторые диагностические методы, обладающие достаточно общей применимостью, которые могут помочь нам уточнить диагноз за пределами этапа обнаружения узкого места. Они по большей части относятся к категориям, перечисленным Беллом и др. (1972) и приведенным в табл. 7.1. В частности, все методы интерпретации результатов измерений и имитационного моделирования, описанные в разд. 2.7, можно использовать для генерации или проверки гипотез в работе по усовершенствованию. Мы теперь покажем, как два метода графического представления данных, использованные для иллюстрации в разд. 2.7.1, могут послужить этой цели.

Профиль использования системы (см. рис. 2.13) является простейшим, но эффективным средством демонстрации временных соотноше-

ний между занятыми и свободными интервалами компонентов системы. Он дает нам в понятной форме некоторую информацию о степени перекрытия периодов работы компонентов. Эта информация, которую невозможно получить из одних лишь значений коэффициентов использования, показывает, чем заняты остальные компоненты, когда один из них свободен. Таким образом, профиль использования системы часто помогает нам понять, почему в нашей системе мультипрограммирование не настолько эффективно, как могло бы быть, или, во всяком случае, почему производительность ниже, чем следовало бы ожидать. Это диагностическое средство популярно, в частности, и из-за легкости получения необходимых данных с помощью аппаратного измерителя. Аналогичные доводы можно привести для *графика Кивиата*. Использование этих двух приемов будет теперь проиллюстрировано тремя примерами, навеянными работой Кокрума и Крокетта (1971).

Пример 7.4. Рассмотрим версию ПРОП-системы с одним ЦП и одним каналом ввода-вывода и предположим, что сеанс измерения дал результаты, приведенные в табл. 2.3. Тогда профиль использования системы можно изобразить, как на рис. 2.13, а соответствующий график Кивиата — как на рис. 2.14. Исследование коэффициентов использования показывает, что эта система имеет незначительную несбалансированность. Профиль показывает также, что степень перекрытия является разумной (около 78% времени, когда занят ЦП, и около 70% времени, когда занят канал). Чтобы уменьшить общее время обработки той же рабочей нагрузки без изменения конфигурации оборудования, мы должны уменьшить время, когда система свободна, или, возможно, попытаться увеличить перекрытие во времени ЦП и канала ввода-вывода. Изучение графика Кивиата на рис. 2.14 подтверждает наше предположение, что в первую очередь следует попытаться уменьшить свободное время всей системы. Таким образом, мы должны сформулировать некоторую гипотезу, чтобы объяснить, почему около 20% общего израсходованного времени система проводит в состоянии ожидания, когда ни один компонент не выполняет никакой работы. Возможны три гипотезы, ни в коей мере друг друга не исключающие:

a) система была пуста во время определенных периодов сеанса измерения;
b) система ожидала действий оператора (например, установки ленты или пакета дисков) или подключения к системе временно отключенного устройства;
c) система ожидала завершения поиска на диске (заметим, что канал ввода-вывода не занят во время поиска).

Каждая из гипотез требует получения дополнительных данных. Профиль на рис. 2.13 не позволяет нам решить, какие из гипотез и в какой мере справедливы. В частности, нам следовало бы извлечь из собранных данных, из учетного журнала за день эксперимента и из прочих системных журналов следующую информацию:

a) количество времени на протяжении сеанса, в течение которого *система была свободна и входная очередь пуста*;
b) количество времени на протяжении сеанса, в течение которого различные устройства *были не готовы*;
c) количество времени, занятого *только поиском* на диске, на протяжении сеанса.

Наибольшее время (или времена) среди только что перечисленных будет указывать на область (или области), где дальнейшее исследование представляется перспективным. В этой области придется формулировать новые гипотезы (почему так велики времена поиска на диске? Почему операторы так медленно устанавливают ленты?), пока не будет найдена наиболее разумная и перспективная терапия. Заметим, что если гипотеза «а» объясняет большую часть времени, когда система свободна, можно прийти к единственно возможному заключению, что либо время сеанса выбрано неудачно, либо емкость системы используется не до конца.

Если мы исследуем возможность увеличения перекрытия ЦП и канала ввода-вывода, то необходимо сформулировать гипотезы о причинах не вполне удовлетворительной величины этого перекрытия в нашей системе. Возможны две причины:

a) плохое планирование заданий: смесь выполняемых заданий не балансируется достаточно хорошо относительно характеристик ресурсов; слишком много заданий, лимитируемых вводом-выводом, дают большой совместный вклад во время, когда занят *только канал*, а слишком много заданий, лимитируемых ЦП, дают большой вклад во время, когда занят *только ЦП*;

b) неэффективные программы: те же проблемы, которые были объяснены выше плохим планированием, могут быть также обусловлены отдельными программами, очень часто выполняемыми на данной установке.

Если обе гипотезы подтверждаются, то они прямо указывают на области, в которых необходимо улучшение. ■

Пример 7.5. Пусть теперь предстоит усовершенствовать систему, описанную в примере 7.1 (см. табл. 7.2). Так как мы уже разносторонне проанализировали модель этой системы, изображенную на рис. 7.5, то можем поинтересоваться, каков же профиль использования этой системы и какие гипотезы можно было бы сформулировать, исходя из него. Этот профиль можно получить, вычисляя степень перекрытия работы различных групп обслуживающих приборов. Это вычисление будет простым, если известны вероятности состояний равновесия. Например, перекрытие между ЦП и барабаном пропорционально сумме вероятностей состояний, в которых как ЦП, так и барабан одновременно имеют ненулевое количество заданий. Для модели на рис. 7.5 вероятность перекрытия ЦП с барабаном равна [по (4.102)]

$$\rho_{12} = \sum_{\substack{\text{все } \mathbf{N} \\ N_1>0 \\ \text{и} \\ N_2>0}} p(N_1,\ N_2,\ N_3) = y_2 \frac{G(N-2)}{G(N)}, \tag{7.18}$$

где y_2 получаем из (4.97). Вывод формулы (7.18), который можно проделать рассуждением того же типа, что был использован для получения выражения ρ_1 в (4.106), оставляем читателю в качестве упражнения.

Аналогично, для остальных вероятностей перекрытия, получим

$$\rho_{13} = y_3 \frac{G(N-2)}{G(N)} = \frac{y_3}{y_2}\rho_{12}, \tag{7.19}$$

$$\rho_{23} = y_2 y_3 \frac{G(N-2)}{G(N)} = y_3\rho_{12} = y_2\rho_{13}, \tag{7.20}$$

$$\rho_{123} = y_2 y_3 \frac{G(N-3)}{G(N)}. \tag{7.21}$$

Вероятности для каждого компонента быть единственным занятым компонентом равны

$$\rho_{10} = \frac{1}{G(N)} = \rho_1 - \rho_{12} - \rho_{13} + \rho_{123}, \tag{7.22}$$

$$\rho_{20} = \frac{y_2^N}{G(N)} = \rho_2 - \rho_{12} - \rho_{23} + \rho_{123}, \tag{7.23}$$

$$\rho_{30} = \frac{y_3^N}{G(N)} = \rho_3 - \rho_{13} - \rho_{23} + \rho_{123}. \tag{7.24}$$

Вывод соотношений, аналогичных (7.18) — (7.24), для модели с центральным обслуживанием с s приборами ($s>3$) является темой задачи 7.6.

На рис. 7.13(a) изображен профиль, полученный из этих соотношений для значений параметров, заданных в табл. 7.3. Заметим, что в этой модели не может быть моментов времени, когда система простаивает. Число заданий в памяти всегда равно N, все обслуживающие приборы всегда действуют, дисковое устройство считается занятым во время поиска. В этом профиле степень перекрытия не слишком неудовлетворительна, а времена, когда занят *только ЦП* или *только барабан*, достаточно малы.

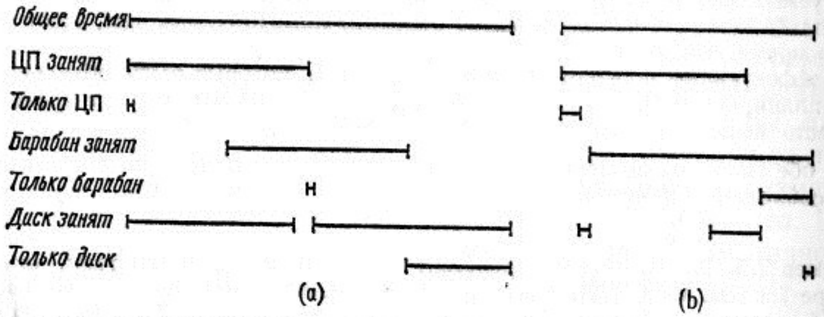


Рис. 7.13. Профили использования системы, полученные на модели рис. 7.5; (a) — при значении параметров из табл. 7.3 и (b) — при тех же значениях параметров, за исключением $b_{12}=0.92$ и $b_{13}=0.03$.

Однако время, когда занят *только диск*, превышает 25% общего времени, и можно сформулировать гипотезу, что иное распределение файлов между диском и барабаном увеличит перекрытия (особенно ρ_{123}).

При построении графика Кивиата для этой двухканальной системы переменные, которые будут изображены на графике, можно выбирать различными способами. Мы выберем коэффициенты использования компонентов ρ_1, ρ_2, ρ_3 и относительное перекрытие каналов $\rho_{23}/\min(\rho_2, \rho_3)$ в качестве «хороших» переменных. Неперекрывающиеся доли этих коэффициентов использования, ρ_{10}/ρ_1, ρ_{20}/ρ_2, ρ_{30}/ρ_3, и часть общего

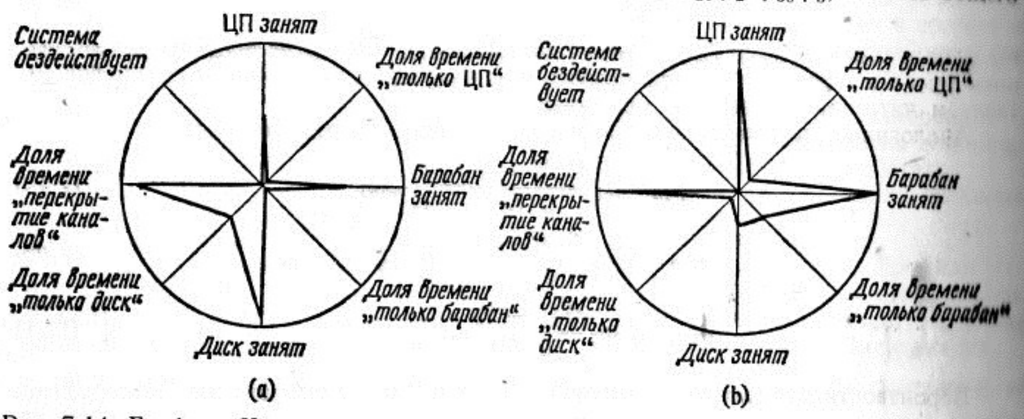


Рис. 7.14. Графики Кивиата для модели рис. 7.5: (a) — со значениями параметров из табл. 7.3 и (b) — с теми же значениями, кроме $b_{12}=0.92$ и $b_{13}=0.03$.

времени, в течение которого система свободна, возьмем в качестве «плохих» переменных. Такой выбор диктуется как желанием избежать переменных, взаимозависимости между которыми слишком очевидны (например, ρ_1 и $1-\rho_1$), так и желанием использовать этот график в диагностических целях. Заметим, что по (7.1) и (7.2), $\overline{T}$ пропорционально ρ_1, а $\overline{t}$ обратно пропорционально, поскольку мы не ожидаем изменения N, b_{11} или μ_1. Кроме того, относительное перекрытие каналов и относительные коэффициенты неперекрывающегося использования выбраны вместо соответствующих абсолютных величин, так как они, по-видимому, более значимы с точки зрения диагностического

использования графика. С учетом этого выбора график Кивиата, соответствующий системе с профилем, приведенным на рис. 7.13(a), изображен на рис. 7.14(a). Читатель может убедиться, что этот график подразумевает те же предположения и гипотезы, что профиль на рис. 7.13(a).

Профиль на рис. 7.13(b) представляет ситуацию в точке максимума $\overline{T}$, достигаемого изменением только b_{12} и b_{13} в табл. 7.3. Неперекрывающееся время обмена с диском практически исчезло. Однако теперь относительно велико время, когда занят *только барабан* (около 20% от всего времени), но барабан быстрее диска. Кроме того, вероятности перекрытия ρ_{13}, ρ_{23}, ρ_{123} уменьшились, но это можно приписать заметному уменьшению ρ_3. На самом деле сравнение профилей нужно делать в терминах времен, а не в терминах коэффициентов использования, раз нашим индексом производительности является $\overline{T}$. Большие значения ρ_1 и ρ_2 помогают обосновать увеличение $\overline{T}$ (которое стало лучше на 53%). Однако из этого профиля не следует, что эта ситуация оптимальна и что никакие изменения b_{12} и b_{13} не могут улучшить $\overline{T}$. Те же выводы можно сделать, исследуя соответствующий график Кивиата, изображенный на рис. 7.14(b). Хотя этот график, конечно, будет признан «лучшим» по сравнению с изображенным на 7.14 (a), его значение ρ_{20}/ρ_2 большинство исследователей оценит как слишком большое, а ρ_3 — как слишком маленькое. Этот очень простой пример должен напомнить нам о рассмотренном в разд. 7.3.2 отличии систем с равномерным использованием от систем, реализующих максимум $\overline{T}$. Нам также не следует забывать, что профили использования системы и графики Кивиата являются средствами диагностики, основанными на отождествлении равномерного использования и максимального перекрытия с максимумом производительности. Мы уже видели, что такое отождествление возможно только в первом приближении. Решение задачи 7.4 даст читателю пример системы с более «приятными» профилем и графиком Кивиата, имеющей меньшую производительность, чем система с менее «симпатичными» характеристиками.

■

Пример 7.6. Нам предстоит изучить систему, имеющую один ЦП, два канала ввода-вывода и восемь внешних накопителей (дисков), по четыре на каждый канал. В табл. 7.5 приведены результаты, полученные при измерениях системы. Соответствующие профиль использования системы и график Кивиата, построенные в соответ-

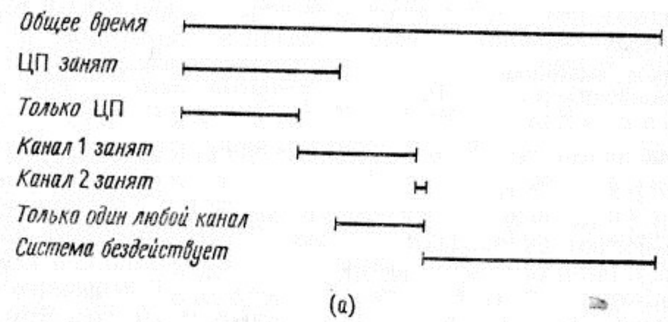
Таблица 7.5

Результаты измерений, проведенных на вычислительной системе

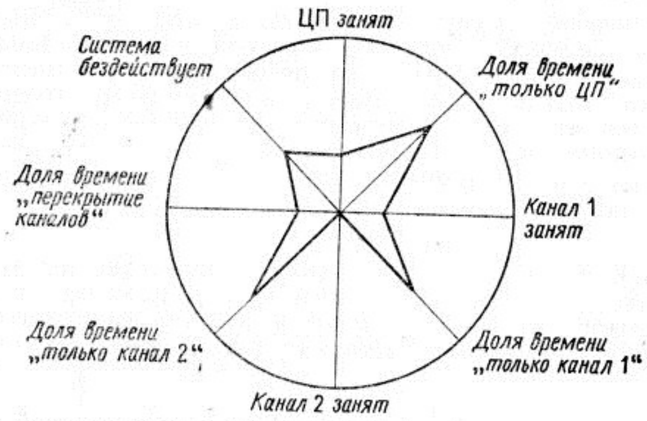
Величина	Описание	Доля общего времени (%)
ρ_1	ЦП занят	33.33
ρ_2	Канал 1 занят	25.00
ρ_3	Канал 2 занят	1.65
ρ_{10}	Только ЦП	24.59
ρ_{23}	Перекрытие каналов	0.41
$1-\rho_{10}-\rho_2-\rho_3+\rho_{23}$	Система свободна	49.17
δ_1	Устройство 1 занято	4.16
δ_2	Устройство 2 занято	2.77
δ_3	Устройство 3 занято	9.73
δ_4	Устройство 4 занято	8.34
δ_5	Устройство 5 занято	0.27
δ_6	Устройство 6 занято	0.83
δ_7	Устройство 7 занято	0.41
δ_8	Устройство 8 занято	0.14

Источник: Кокрум и Крокетт (1971), с разрешения AFIPS, Inc.

ствии с решением, принятым в примере 7.5, представлены на рис. 7.15. Как профиль, так и график указывают на некоторые очевидные типы проблем, в особенности на большое время простоя системы, несбалансированность коэффициентов использования каналов и на большое относительное время неперекрывающейся работы каждого компонента. Гипотезы, которые можно сформулировать для объяснения избытка времени, когда система свободна, были уже перечислены и обсуждены в примере 7.4.



(a)



(b)

Рис. 7.15. Профиль использования системы (a) и график Кивиата (b), соответствующие результатам измерений в табл. 7.5. Так как соответствующие данные оказались недоступны, было предположено, что работа канала 2 не перекрывается с работой ЦП.

Остальные два главных симптома, упомянутые выше, по всей видимости, указывают на то, что привязка устройств внешней памяти к каналам сделана неудачно. Измерения, приведенные в табл. 7.5, подтверждают эту гипотезу. Устройства с первого по четвертое, наиболее используемые, присоединены к каналу 1, тогда как канал 2 управляет остальными четырьмя устройствами, коэффициенты использования которых приблизительно на порядок ниже. Следовательно, перекоммутация связей между устройствами ощутимо улучшит ситуацию. Мы вернемся к этому примеру в разд. 7.4.

■

Как только гипотеза сформулирована, ее следует проверить. Некоторые гипотезы относительно легко поддаются проверке. Как и случае гипотезы о неудачном присоединении устройств в примере 7.6, проверка может иногда сводиться к изучению дополнительных экспериментальных данных. Часто эти данные не являются легко доступными и требуют проведения специально спланированного эксперимента для своего получения, однако сделать это можно за счет небольшого увеличения стоимости работы по усовершенствованию. В других случаях проверка гипотезы измерительными методами могла бы потребовать намного больших усилий и расходов. Это верно, например, для гипотез, согласно которым недостаточная скорость одного из компонентов оборудования является главной причиной возникновения узкого места. Проверка справедливости такой гипотезы прямым увеличением скорости этого компонента (и, возможно, некоторых других компонентов также) обычно нереальна. Однако, если экспериментатор использует исполняемую искусственную модель рабочей нагрузки для управления системой, возможна до некоторой степени косвенная, но все же экспериментальная проверка, особенно если модель состоит из набора синтетических заданий. Например, уменьшение запроса на обслуживание компонентом, предположительно создающим узкое место, можно было бы интерпретировать как в некотором смысле эквивалентное увеличению скорости этого компонента (см. задачу 7.2).

Проверка гипотезы может быть выполнена с намного меньшими ограничениями, если воспользоваться имитационной моделью улучшаемой системы. Все виды изменений, включая физически нереализуемые, можно испытать на хорошо разработанной имитационной модели. Единственное практическое ограничение — это стоимость прогонов модели. Широкое исследование по обнаружению и устранению узких мест было описано Нильсеном (1967). Обсудим вкратце некоторые аспекты этой работы.

Пример 7.7. Моделируется система с разделением времени IBM 360/67. Первоначальная конфигурация аппаратуры состоит из центрального процессора, 3 модулей основной памяти (каждый по 256K байт) и 6 каналов ввода-вывода. Эти каналы управляют следующими внешними накопителями: 1 страничный барабан, 1 дисковое устройство с 8 модулями для страничных обменов, еще одно такое же устройство для хранения файлов, 2 меньших дисковых устройства также для хранения файлов и 1 диалоговый терминал. Внешние устройства, такие, как устройства ввода перфокарт, печати и лентопротяжные устройства, хотя и являются частью конфигурации, не моделировались. Модель рабочей нагрузки состоит из диалоговых задач, выбираемых из 10 классов, характеризующихся широким спектром запросов на вычислительные ресурсы и страницы памяти. Пакетные задания и задания реального времени не включались в рассмотрение, однако легко превратить часть диалоговых задач в пакетные исключением взаимодействия с терминалами. Модель рабочей нагрузки также можно изменять различными способами, наиболее очевидным из которых является изменение числа терминалов, выполняющих задачи каждого типа.

Первичным индексом производительности является коэффициент использования ЦП, который при определенных предположениях (см. пример 7.1) пропорционален средней пропускной способности. Точнее, относящаяся к накладным расходам работа ЦП, которая тратится на назначение ресурсов, обработку страничных отказов и прерываний и выполнение остальных системных функций, не дает непосредственного вклада в $\bar{T}$. Таким образом, мы на самом деле интересуемся величинами двух составляющих ρ_1: пользовательским коэффициентом ρ_{1p} и системным коэффициентом ρ_{1s}. Из остальных коэффициентов использования интерес представляют коэффициент использования страничного барабана (ρ_2), страничного диска (ρ_3) и дисков, используемых для хранения файлов (ρ_4). Дополнительные индексы, такие, как время ответа, частота замены страниц (число переданных страниц в единицу времени) и средние длины очередей также включены в рассмотрение из-за их диагностической ценности.

Моделирование системы в ее первоначальной конфигурации и с ее первоначальной рабочей нагрузкой дает следующие результаты: $\rho_{1p}=0.12$, $\rho_{1s}=0.39$, $\rho_2=0.07$, $\rho_3=0.80$ и $\rho_4=0.14$—0.32. Такой уровень производительности, конечно, неудовлетворителен. Краткое описание некоторых из многих циклов *гипотеза — проверка — результат*, рассмотренных Нильсеном (1967), приведено ниже.

Гипотеза: Большой период времени, когда ЦП простаивает, обусловлен слишком малой нагрузкой.

Проверка: Превратим 12 диалоговых задач с наибольшим запросом времени ЦП в пакетные.

Результат: $\rho_{1p}=0.14$, $\rho_{1s}=0.43$, $\rho_2=0.10$, $\rho_3=0.58$, $\rho_4=0.14$—0.58. Гипотеза должна быть отвергнута.

Гипотеза: Производительность снизилась из-за наличия узкого места. Поскольку ρ_3 довольно велико, узкое место создается страничным диском.

Проверка: Устраним необходимость в страничном диске, уменьшив рабочую нагрузку до 12 пакетных заданий, лимитируемых вычислениями, с минимальным количеством запросов на обмен страницами.

Результат: $\rho_{1p}=0.43$, $\rho_{1s}=0.35$, $\rho_2=0.08$, $\rho_3=0.00$, $\rho_4=0.22$. Гипотеза справедлива.

Гипотеза: Можно получить удовлетворительную производительность, добавив три дисковых страничных устройства, каждое из которых подсоединено к отдельному каналу.

Проверка: Расширить модель системы и осуществить ее прогон при первоначальной рабочей нагрузке.

Результат: $\rho_{1p}=0.20$, $\rho_{1s}=0.46$, $\rho_2=0.12$, $\rho_3=0.08$—0.12, $\rho_4=0.30$—0.50. Гипотеза подтвердилась только частично, так как производительность пока еще далека от приемлемой.

Гипотеза: Есть другое узкое место, в данный момент созданное дисками для хранения файлов.

Проверка: Свести к нулю времена доступа и пересылки у дисков для хранения файлов.

Результат: $\rho_{1p}=0.31$, $\rho_{1s}=0.69$, $\rho_2=0.23$, $\rho_3=0.13$—0.17, $\rho_4=0.00$. Гипотеза справедлива, но проблема накладных расходов стала непреодолимой.

Здесь исследователи столкнулись с двумя проблемами. Во-первых, нужно решить проблему накладных расходов. Тщательное изучение того, как работает операционная система, наряду с уже имеющимся опытом или другими методами (см. табл. 7.1) поможет сформулировать гипотезы в этой области. Другая проблема относится к практическому устранению обнаруженных узких мест. Подход, использованный для проверки последней гипотезы (а именно, предположение о бесконечно быстрых дисках для хранения файлов), не реалистичен. Аналогично, добавление трех каналов и трех восьмимодульных дисковых устройств для устранения узкого места на страничном диске, хотя и осуществимо, но не экономично. Таким образом, следует формулировать реалистические гипотезы об усовершенствованиях. Они включают в себя, например, изменение алгоритмов управления диском, чтобы сделать его более эффективным, и замену страничного дискового устройства вторым страничным барабаном. Следовательно, подходящая имитационная модель может быть весьма полезна и при планировании этапа терапии в работе по усовершенствованию. ■

7.3.4. Статистический метод диагностики

В разд. 7.3.1, методы, основанные на последовательном устранении узких мест, были признаны полезными для повышения производительности. Соответственно обнаружение узких мест было представлено как основная задача на этапе диагностики работы по усовершенствованию. Формальное определение узкого места, данное в разд. 7.3.2 [см. (7.3)],

и анализ простой модели мультипрограммной системы показали, что, хотя балансировка системы или выравнивание коэффициентов использования ее компонентов обычно не максимизируют ее производительность, они очень часто служат удобными целями для усовершенствования. Концепцию, на которой основано определение (7.3), можно применить и к другим индексам производительности, а не только к $\overline{T}$. В задаче 7.3 читателю предлагается распространить это определение на случай, когда первичным индексом производительности служит среднее время прохождения $\overline{t}$. Однако это возможно лишь тогда, когда индекс — единственная скалярная переменная. Как определить узкие места, если индекс производительности — вектор или статистическое распределение? Возможное определение неявно подразумевалось Бейлнером (1973) в его предложении применения к задачам улучшения производительности *метода хорошего баланса*, описанного в разд. 3.3.3.

Пусть рабочая нагрузка или ее исполняемая модель состоит из n заданий, и пусть вектор $(t_1, t_2, \ldots, t_n)$ их времен прохождения будет нашим индексом производительности. Предположим, что *требуемое время прохождения* каждого задания каким-то образом известно. Например, мы можем выбрать в качестве требуемого времени прохождения t_i^* для i-го задания минимальное возможное значение t_i, т. е. время, необходимое для выполнения этого задания в системе при однопрограммном режиме. Таким образом, если мы знаем вектор $(t_1^*, t_2^*, \ldots, t_n^*)$ для заданной конфигурации оборудования и программного обеспечения, то мы можем измерить вектор $(t_1, t_2, \ldots, t_n)$ и вычислить вектор отклонения $\mathbf{D}=(d_1, d_2, \ldots, d_n)$, где $d_i=t_i-t_i^*$.

Каждое задание из рабочей нагрузки должно количественно характеризоваться набором из v параметров $(w_1, w_2, \ldots, w_v)$. Например, эти параметры могут означать запросы ресурсов на каждое задание. Предположим для простоты, что все w измерены в одинаковых единицах (в секундах). Отклонение $d=t-t^*$, значения которого равны $d_1, d_2, \ldots, d_n$, является функцией этих параметров. Дадим следующее определение узкого места, которое можно в некотором смысле рассматривать как обобщение (7.3): узкое место создается j-м ресурсом ($1\leqslant j\leqslant v$), если для всех h, таких, что $1\leqslant h\leqslant v$, $h\neq j$,

$$\frac{\partial d}{\partial w_j} > \frac{\partial d}{\partial w_h}. \tag{7.25}$$

Аналогично, система называется сбалансированной, если для всех j и h

$$\frac{\partial d}{\partial w_j} = \frac{\partial d}{\partial w_h}. \tag{7.26}$$

Однако d, вообще говоря, является функцией со случайным поведением. Постулируя, что d можно представить суммой случайных компонентов и детерминированного тренда, мы можем попытаться аппроксимировать этот тренд уравнением регрессии. Линейное уравнение регрессии первого порядка вполне подходит для наших целей об-

наружения узких мест. Коэффициенты a_0 и a_h в уравнении

$$d = a_0 + \sum_{h=1}^{v} a_h w_h \qquad (7.27)$$

можно получить из t, t^* и w_h для каждого задания, как показано в разд. 2.7.3. Если уравнение (7.27) значимо (в статистическом смысле), то те коэффициенты a_h, которые значимы, могут быть использованы для обнаружения узких мест в качестве частных производных в определении (7.25). Однако условие баланса (7.26) должно быть заменено условием незначимости уравнения (7.27). В этом случае отклонения сбалансированы в том смысле, что любые запросы на ресурсы влияют на отклонения примерно в одинаковой степени. Другими словами, существующие отклонения нельзя более объяснить нехваткой определенного ресурса. Они обусловлены другими причинами, в частности ожиданиями в очередях мультипрограммных систем. Таким образом, к этому случаю можно применить метод последовательного устранения узких мест, причем целью здесь будет получение незначимого уравнения регрессии путем последовательных изменений системы. На эти изменения указывают относительные величины значимых коэффициентов в (7.27). Следует тщательно рассмотреть возможность того, что предположения, от которых зависит справедливость F-теста, по всей вероятности, не будут выполнены (см. разд. 2.7.2). В таких случаях всегда рекомендуется придерживаться консервативной линии поведения, т. е. выбирать меньшие вероятности ошибок, чем допустимые.

7.4. ТЕРАПИЯ

7.4.1. Настроечная терапия

Операционные системы создаются для большого числа разнообразных установок с различными конфигурациями оборудования и различными рабочими нагрузками. Это диктуется очевидными экономическими преимуществами. Поэтому удовлетворительный уровень производительности может быть достигнут только в том случае, если операционная система приспособлена к характеристикам данной установки. В то время как конфигурация оборудования данной установки известна заранее и вполне стабильна во времени, фактическая рабочая нагрузка может быть предсказана только приблизительно и на основе весьма общих соображений. К тому же она меняется во времени. Таким образом, подгонка операционной системы к рабочей нагрузке должна производиться на работающей системе и, вообще говоря, должна повторяться неоднократно в течение времени существования системы. Эта операция подгонки называется *настройкой* (термин, удачно заимствованный из сферы музыкальных инструментов). Мы будем использовать здесь этот термин в несколько более широком смысле для обозначения всех методов терапии, улучшающих производительность, но не связанных с изменениями конфигурации оборудования. Изменения в подключении оборудования, не вызывающие добавления какого-либо оборудования, также будут считаться настроечной терапией.

Таблица 7.6

Главные типы методов настроечной терапии

Изменяемая величина	Примеры
Параметры операционной системы	Степень мультипрограммирования Размер буфера Квант времени Определения классов приоритета Размер страницы
Алгоритмы управления ресурсами	Алгоритмы планирования (планирование заданий, ЦП, барабана, диска) Алгоритмы адресации (методы доступа) Алгоритмы управления памятью (распределение, замещение, загрузка, попеременная загрузка заданий) Алгоритмы управления внешними устройствами Ручные процедуры (установка и снятие дисковых пакетов и лент, ручное планирование заданий, ручная попеременная загрузка заданий)
Подсоединения	Соединения устройств с каналами Соединения устройств с магистралями
Размещение информации	Размещение файлов в иерархической памяти Резидентные модули операционной системы
Производительность программ	Накладные расходы операционной системы Время выполнения и место в памяти, необходимое для часто используемых программ (компиляторы, редакторы, системы баз данных, программы сортировки) Локальность обращений к памяти
Политика цен	Льготы для пользователей в периоды облегченной нагрузки Штрафы за неэффективность программ Штрафы за использование перегруженных ресурсов
Политика приема заданий	Исключение определенных типов заданий или определенных пользователей Ограничение нагрузки

Наиболее важные типы изменений в рамках настройки и некоторые примеры каждого типа перечислены в табл. 7.6. Эта таблица показывает, насколько широк спектр изменений, не затрагивающих оборудования, но оказывающих заметное положительное воздействие на производительность. Частные исследования, иллюстрирующие реа-

лизацию и результаты таких изменений на множестве систем, описываются во многих публикациях. Большинство примеров, перечисленных в табл. 7.6, представляют классы методов терапии; эти примеры могут быть значительно расширены.

Глядя на табл. 7.6, читатель поймет, насколько может оказаться полезной для оценщиков системы тщательная диагностика. Подробный диагноз становится существенным, когда исследователь полагает, что многие изменения, перечисленные в табл. 7.6, являются или могут оказаться слишком дорогими, несмотря на то что они не требуют дополнительного оборудования. В частности, изменения алгоритмов автоматического управления ресурсами, политики цен и производительности программ обычно требуют изменения существующих программ, а также написания новых программ и их согласования с существующими. Эта дорогая, требующая времени и порождающая ошибки работа может быть предпринята, только если ее целесообразность будет определена с большой степенью уверенности. Остальные изменения, например изменения параметров операционной системы, взаимосвязей, размещения информации, требуют некоторых изменений базы данных операционной системы. Хотя эти операции являются быстрыми и дешевыми, их следует выполнять весьма тщательно, поскольку результат ошибок может оказаться катастрофическим. Кроме того, большинство изменений может вызвать нарушения в работе и непредвиденные реакции со стороны коллектива пользователей. Эти реакции и нарушения могут на время или навсегда свести на нет ожидаемые улучшения. Иногда с учетом приведенных выше соображений разумнее не производить настроечную терапию, предложенную на этапе диагностики, а выбрать другое направление действий. В остальных случаях нам приходится выбирать между несколькими возможными методами терапии.

Все эти решения должны основываться на критерии стоимость — производительность». Окупится ли планируемое усовершенствование? Какой из возможных методов терапии даст наибольшее вероятное улучшение при наших затратах?

Аспект производительности этого критерия требует, чтобы для прогнозирования улучшения производительности можно было применять относительно простые, недорогие и надежные методы. Если изменение легко и быстро выполнимо, а также дешево, то лучший метод обычно экспериментальный. Однако, как отмечено выше, большинство изменений, перечисленных в табл. 7.6, не принадлежит к этому типу. Таким образом, если методы измерения нельзя применить для анализа производительности, остается пользоваться методами имитационного моделирования и аналитическими методами. Каждый из них имеет свои преимущества и недостатки, рассматриваемые в разд. 7.5. В примере 7.7 кратко проиллюстрировано применение моделирования к задаче прогнозирования производительности в контексте работы по усовершенствованию. Теперь мы приведем несколько примеров использования аналитических методов для этой же цели.

Пример 7.8. В результате анализа данных табл. 7.5 (см. рис. 7.15 и пример 7.6) выдвинуто предложение об изменении подсоединения устройств к каналам. В частности, чтобы лучше распределить общую нагрузку каналов, мы можем подсоединить диски 2, 3, 5 и 7 к каналу 1, а диски 1,4, 6 и 8 к каналу 2. Как можно оценить улучшение производительности в результате такого изменения? Приближенный, но часто достаточно точный метод, предложенный Кокрумом и Крокеттом (1971), основан на *уравнении времени системы*, полученном из профиля использования системы. Уравнение времени системы — это уравнение, которое выражает общее время в виде суммы членов, соответствующих компонентам системы. Например, исходя из профиля на рис. 7.15 (а), можно написать

$$t_{\text{общ}} = t_{\text{ЦП занят}} + t_{\text{только любой канал}} + t_{\text{система бездействует}}. \tag{7.28}$$

Используя обозначения из табл. 7.5, получим

$$t_{\text{ЦПзанят}} = \rho_1 t_{\text{общ}},\ t_{\text{только любой канал}} = [\rho_2 + \rho_3 - \rho_{23} - (\rho_1 - \rho_{10})]\, t_{\text{общ}},\ t_{\text{система бездействует}} = (1 + \rho_{10} - \rho_2 - \rho_3 + \rho_{23})\, t_{\text{общ}}. \tag{7.29}$$

Читатель может проверить равенство (7.28), подставляя в него (7.29).

Теперь мы предположим, что изменение подсоединения устройств к каналам не влияет на $t_{\text{ЦП занят}}$ и $t_{\text{система бездействует}}$. Это предположение вполне разумно для $t_{\text{ЦП занят}}$, а также в первом приближении и для $t_{\text{система бездействует}}$. Таким образом, нужно определять лишь изменения $t_{\text{только любой канал}}$. Новые коэффициенты использования можно вычислить по данным табл. 7.5; получим $\rho_2' = \delta_2 + \delta_3 + \delta_5 + \delta_7 = 13.18\%$ и $\rho_3' = \delta_1 + \delta_4 + \delta_6 + \delta_8 = 13.47\%$. Новую вероятность перекрытия каналов ρ_{23}' придется оценивать по теоретико-вероятностным соображениям, так как нет никакой дополнительной информации. Заметим, что δ_i — вероятность активного состояния i-го устройства на протяжении сеанса измерения, и мы полагаем, что она останется неизменной после изменения подключения устройств. Предполагая активности устройств статистически независимыми, получим вероятность перекрытия в виде

$$\rho_{23}' = (\delta_2 + \delta_3 + \delta_5 + \delta_7)(\delta_1 + \delta_4 + \delta_6 + \delta_8) = \rho_2' \rho_3' = 1.77\%, \tag{7.30}$$

а из (7.29) мы имеем

$$t'_{\text{только любой канал}} = [\rho_2' + \rho_3' - \rho_{23}' - (\rho_1 - \rho_{10})]\, t_{\text{общ}}. \tag{7.31}$$

Заметим, что предположение о независимости подтверждается результатом оценки ρ_{23} произведением $\rho_2\rho_3$, которое равно 0.415% (см. табл. 7.5). Однако изучение модели с центральным обслуживающим прибором в примере 7.5 не подтверждает это предположение (см., например, (7.20)).

Уравнение времени системы (7.28) дает

$$\frac{t_{\text{общ}} - t'_{\text{общ}}}{t_{\text{общ}}} = \rho_2 + \rho_3 - \rho_{23} - \rho_2' - \rho_3' + \rho_{23}' = \rho_{23}' - \rho_{23} = 1.36\%.$$

Таким образом, изменение подключения устройств, как ожидается, уменьшит общее время на 1.36%. Это приблизительно равно ожидаемому улучшению средней пропускной способности. ■

Пример 7.9. Теперь изучим с помощью модели с центральным обслуживающим прибором влияние изменений размещения информации на производительность. Эта работа по усовершенствованию была выполнена Хьюзом и Мо (1973) для машины UNIVAC 1108 с операционной системой EXEC 8. Данная конфигурация включала один ЦП (обслуживающий прибор Σ_1), высокоскоростной барабан FH432 (обслуживающий прибор Σ_2), среднескоростной барабан FH880 (обслуживающий прибор Σ_3), медленный барабан FASTRAND (обслуживающий прибор Σ_4), лентопротяжные устройства (обслуживающий прибор Σ_5). Система моделировалась сетью с центральным обслуживанием с пятью обслуживающими приборами с очередями. Хьюз и Мо

определили пропускную способность как число циклов «ЦП — внешнее устройство» при выполнении N заданий в единицу времени и при $b_{11}=0$. Однако эти отличия от нашего предыдущего обсуждения модели центрального обслуживания здесь не важны, так как мы будем оценивать влияние изменений на коэффициенты использования различных обслуживающих приборов, а не на индексы типа T или $\bar{t}$.

В табл. 7.7 приведены коэффициенты использования (ρ_i) и общие количества запросов (n_i) для пяти обслуживающих приборов, представленных в модели. Эти зна

Таблица 7.7

Оценка результатов изменения размещения информации (см. пример 7.9)

Номер обслуживающего прибора, i	Ресурс	Коэффициент использования до изменения, ρ_i	Число запросов до изменения, n_i	Оценка числа запросов после изменения, $\hat{n}_i'$	Оценка коэффициента использования после изменения, $\hat{\rho}_i'$	Измеренный коэффициент использования после изменения, ρ_i'
1	ЦП	0.62	67024	67024	0.77	0.74
2	Барабан FH432	0.26	33626	33626	0.32	0.29
3	Барабан FH880	0.14	7351	12788	0.30	0.27
4	Барабан FASTRAND	0.90	14763	9326	0.71	0.68
5	Магнитная лента	0.18	11284	11284	0.22	0.22

Источник: Хьюз и Мо (1973).

чения были измерены во время эксперимента, в котором система управлялась набором эталонов. Так как коэффициент ρ_4 использования барабана FASTRAND намного выше остальных, то этот барабан, по-видимому, создает узкое место. Лучшее распределение нагрузки между устройствами можно было бы получить, перенеся часть запросов с FASTRAND на барабан FH880, который нагружен слабо. Большое количество обращений к FASTRAND обусловлено тем, что он используется для буферизации информации от медленных периферийных устройств, например от устройств ввода перфокарт, и для буферизации информации, выводимой на медленные устройства, например карточные перфораторы и устройство печати. Передача этой функции буферизации более быстрому барабану FH880 приводит к уменьшению n_4 на 5437 запросов. Предполагая, что мы можем как-то оценить такое уменьшение, не делая слишком большой ошибки, мы получим новые числа запросов ($\hat{n}_i'$), приведенные в табл. 7.7. Эти числа были определены в предположении, что изменение не влияет на n_1, n_2 и n_5 и что все запросы, не поступающие на FASTRAND, поступают на барабан FH880. Кроме того, предполагается, что (экспоненциальные) распределения времен обслуживания и их средние ($1/\mu_i$) для всех обслуживающих приборов при этом не изменяются.

Проблема использования модели центрального обслуживания в данном случае возникает потому, что степень мультипрограммирования N в этой системе не остается постоянной. Чтобы оценить среднее значение N на протяжении эксперимента до изменения, мы можем воспользоваться этой моделью для построения кривой $\rho_1(N)$, которая похожа на кривые рис. 7.9, а затем определить N по измеренному значению ρ_1 (которое в табл. 7.7 равно 0.62). Точки такой кривой, соответствующие целым значениям N, можно вычислить с помощью метода, описанного в разд. 4.3.5, заметив, что значения

$$y_i=\frac{\rho_i}{\rho_1} \quad (i=2,\ \ldots,\ 5) \tag{7.32}$$

можно непосредственно получить из табл. 7.7. Читатель, который пожелает построить кривую $\rho_1(N)$, может получить интерполяцией, что значению $\rho_1=0.62$ соответствует (приблизительно) $N=4.5$. Таково, следовательно, среднее значение степени мультипрограммирования $\bar{N}$, которое мы считаем не подверженным изменениям.

Заметим, что вероятности запросов b_{1i} можно записать в виде

$$b_{1i}=\frac{n_i}{n_1} \quad (i=2,\ \ldots,\ 5) \tag{7.33}$$

и что, согласно уравнению (4.97), y_i пропорционально b_{1i}. Таким образом, поскольку n_1, n_3 и n_5 не подвержены изменениям, то после изменения получим

$$\hat{y}_1'=y_1,\ \hat{y}_2'=y_2,\ \hat{y}_3'=\frac{\hat{n}_3'}{n_3}y_3,\ \hat{y}_4'=\frac{\hat{n}_4'}{n_4}y_4,\ \hat{y}_5'=y_5. \tag{7.34}$$

Затем с помощью (7.34) можно вычислить новую кривую $\hat{\rho}_1'(N)$ и определить ее интерполяцией для $N=4.5$. По значению $\hat{\rho}_1'$, которое оказывается равным приблизительно 0.77, мы можем вычислить все остальные коэффициенты использования с помощью уравнений (7.32) и (7.34). Эти коэффициенты использования $\hat{\rho}_i'$ приведены в табл. 7.7. Сравнение их с измеренными значениями коэффициентов использования после произведенного изменения показывает, что можно получить хорошую точность даже с помощью такого грубого подхода, как использованный в этом примере. ■

Пример 7.10. Теперь мы изучим влияние на производительность системы с виртуальной памятью некоторых изменений в алгоритме управления памятью или в поведении программы. В конце нашего обсуждения в примере 7.3 мы пришли к выводу, что если изменения интенсивности страничных отказов, вызванные изменениями алгоритма управления, поддаются оценке, то для приблизительного определения соответствующих изменений $\bar{T}$ и $\bar{t}$ можно воспользоваться моделью центрального обслуживания (см., например, кривые на рис. 7.12). В примере 7.3 предполагалось, что все задания с точки зрения обращений к памяти в среднем ведут себя одинаково, т. е. значения коэффициента a и показателя степени k в функции времени жизни были одинаковы для всех заданий. В этой ситуации было естественно сделать размеры областей памяти одинаковыми и равными M/N. Однако на практике все программы различны, и нам следовало бы ожидать повышения производительности, если бы приняли стратегию управления переменной памятью, по которой каждому заданию отводится область памяти различного размера. Для простоты предположим, что все программы имеют одинаковое значение k, но различные значения a. Область памяти, отведенная каждому заданию, будет сохраняться постоянной в течение всего времени нахождения задания в памяти, а завершенное задание будет замещаться новым заданием, имеющим такую же функцию жизни. Эти предположения, разумеется, крайне грубы, но наша цель состоит в иллюстрации использования простых аналитических средств при оценке результатов изменения управления памятью.

В нашей системе степень мультипрограммирования постоянна, и рабочая нагрузка характеризуется N коэффициентами $a_1, a_2, \ldots, a_N$ и значением k. Среднее число страничных отказов, порождаемых i-м заданием, согласно (7.12), равно

$$\bar{n}_{\text{бр}\,i}=\frac{\bar{L}}{\bar{v}_{\text{цп}}\bar{e}}=\frac{\bar{L}}{a_i m_i^k}. \tag{7.35}$$

Условие равенства чисел страничных отказов для всех заданий, очевидно, $\bar{e}_i=\bar{e}_j$ для всех i, j. Это условие можно записать в виде

$$m_j=\frac{a_i^{1/k}m_i}{a_j^{1/k}}, \tag{7.36}$$

который, с учетом ограничения

$$\sum_{j=1}^{N} m_j \leqslant M \tag{7.37}$$

и в частности при знаке равенства в (7.37), принимает вид

$$m_j = \frac{M}{\sum_{i=1}^{N} a_i^{-1/k}} a_j^{-1/k}. \tag{7.38}$$

Таким образом, если каждое задание получает объем памяти, обратно пропорциональный $(1/k)$-й степени коэффициента a_j, индивидуальные интенсивности страничных отказов равны между собой и для оценки $\bar{T}$ и $\bar{t}$ можно воспользоваться подходом из примера 7.3. Согласно (7.14) и (7.38), общая средняя интенсивность страничных отказов в этом случае равна

$$\bar{f} = \frac{\bar{v}_{\text{цп}} N}{M^k} \left(\sum_{i=1}^{N} a_i^{-1/k} \right)^k. \tag{7.39}$$

Например, пусть для системы, которая анализировалась в примере 7.3, $N=3$, $M=160$ страниц, $k=2$, $a_1=25$, $a_2=2.77$, $a_3=6.25$ страниц$^{-k}$, а $\bar{v}_{\text{цп}}=10^6$ команд/с. Из (7.39) получаем $\bar{f}=169$ страничных отказов/с, а из рис. 7.12 $\bar{T}=2.58$ задания/с и $\bar{t}=1.16$ с.

Что произойдет, если размер области памяти, отведенной каждому заданию, не удовлетворяет условию (7.38)? Средние числа страничных отказов $\bar{n}_{\text{бр}i}$ будут отличаться друг от друга [см. (7.35)], а каждое задание будет иметь свое распределение времен ЦП или по крайней мере свое значение μ_1. Из табл. 7.3 видно, что μ_1 зависит от значения $\bar{n}_{\text{цп}}$, которое является функцией $n_{\text{бр}}$, как показано в (7.13). Таким образом, наша модель более неприменима. Однако мы еще можем воспользоваться ею, чтобы получить качественные результаты, особенно если интенсивности страничных отказов у программ отличаются не слишком резко и при условии, что мы не слишком полагаемся на эти результаты. Предположим, что память нашей системы разделена поровну между тремя заданиями. Тогда

$$\bar{f} = \bar{v}_{\text{цп}} \sum_{j=1}^{N} \frac{1}{a_j m_j^k} = \bar{v}_{\text{цп}} \frac{N^k}{M^k} \sum_{j=1}^{N} a_j^{-1} \tag{7.40}$$

дает $\bar{f}=197$ страничных отказов/с. Разделение памяти поровну увеличивает общую среднюю интенсивность страничных отказов в этом случае приблизительно на 16%. Если бы были применимы результаты, полученные в примере 7.3 с помощью модели центрального обслуживания, мы могли бы получить из рис. 7.12 новые значения индексов производительности: $\bar{T}=2.45$ задания/с, $\bar{t}=1.22$ с. Следовательно, если бы стратегия разделения поровну была заменена стратегией разделения на области переменного размера, основанной на (7.38), то и $\bar{T}$, и $\bar{t}$ в этом случае, по-видимому, улучшатся приблизительно на 5%. Легко доказать, что (7.38) дает такое разделение на области, которое при сделанных предположениях минимизирует общую среднюю интенсивность страничных отказов.

Только что проиллюстрированный подход можно также использовать для оценки выигрыша от повышения производительности программы. Предположим, что в системе, описанной в примере 7.3, при $N=5$ мы улучшили обработку ссылок в компиляторе, например спроектировав и написав его заново. Предположим также, что компилятор используется столь интенсивно, что его можно считать постоянно находящимся в основной памяти, и предположим, что все программы до изменения давали

одинаковую интенсивность страничных отказов. Насколько повысится производительность системы, если интенсивность страничных отказов компилятора уменьшится на 50%?

Общая средняя интенсивность страничных отказов до изменения может быть вычислена по формуле (7.14): $\bar{f}=781$ страничный отказ/с. Упомянутое выше снижение интенсивности страничных отказов вызовет уменьшение $\bar{f}$ на 10% до значения 703 страничных отказа/с. Из рис. 7.12 мы получим соответствующие значения $\bar{T}$ и $\bar{t}$. $\bar{T}$ увеличится приблизительно на 10% от 1.62 до 1.79, а $\bar{t}$ уменьшится приблизительно на 9% с 3.07 до 2.79. Взгляд на кривые рис. 7.12 убедит читателя, что относительные улучшения, вызванные повышением производительности данной программы, зависят от начальной точки, т. е. от значений N и $\bar{f}$. Они также, очевидно, зависят от ситуации в системе (в смысле узких мест). Читатель может убедиться в том (см. задачу 7.13), что если система не сбалансирована из-за наличия узкого места, созданного диском, улучшение производительности программы или стратегии управления памятью оказывает минимальное воздействие на $\bar{T}$ и $\bar{t}$. Тот же вывод справедлив, если страничный барабан настолько перегружен, что наше уменьшение $\bar{f}$ недостаточно для снятия этой перегрузки. Это очевидно из рис. 7.12 для очень больших значений $\bar{f}$. ■

В связи с настроечной терапией возникает важный вопрос, должна ли она рассматриваться на очередном этапе усовершенствования до или после развития системы. Поскольку развитие включает выбор и приобретение новых компонентов оборудования, разумно предположить, что возможность настройки системы должна быть рассмотрена в первую очередь. Однако, как мы уже отмечали ранее в этом разделе, настройка не обязательно окажется дешевле развития, несмотря на то что некоторые из типов изменений, перечисленных в табл. 7.6, обычно довольно дешевы. Кроме того, результаты измерений могут убедить оценщика в том, что снижение производительности настроенной системы обусловлено скорее увеличением рабочей нагрузки, нежели изменением ее характера. В этих случаях перенастройка системы, по всей вероятности, не внесет существенного вклада в решение проблемы, и следует сразу заниматься развитием системы. После развития системы, разумеется, потребуется настройка.

Но как определить, когда следует прекратить попытку настройки или когда ее даже не следует предпринимать? В этом разделе мы уже рассмотрели несколько простых примеров прогноза усовершенствования. Если все возможные изменения, которые мы рассмотрели, оказываются безрезультатными с точки зрения предсказуемых приемлемых по стоимости усовершенствований, мы должны сделать вывод, что настроечная терапия бесполезна для нашей системы. Это вполне может произойти, если система находится в точке оптимума производительности или близка к ней. Как можно определить, что дело обстоит именно так, без исчерпывающего испытания всех возможных изменений и оценки или измерения результатов этих изменений? Разумеется, эту проблему можно было бы решить, если бы были известны оптимальные значения индексов производительности для заданной конфигурации оборудования и заданной рабочей нагрузки. Мы видели в

начале разд. 7.3.3, что эта информация могла бы оказаться также крайне полезной как симптом недостаточной производительности. Однако даже для самых простых и самых распространенных индексов (как, например, $\overline{T}$) доступны только очень грубые методы. Например, формула (7.15) с $\rho_{1\,max}=1$ часто используется для вычисления емкости системы. В гл. 6 результирующее значение $\overline{T}_{max}$ несколько раз упоминалось в качестве *мощности обработки информации*, несмотря на то что в разд. 6.3.3 приведены другие элементарные количественные определения этого же понятия.

Теперь мы покажем, как можно использовать аналитические методы для оценки емкости данной конфигурации оборудования при заданной рабочей нагрузке. Это очень простой шаг. Если система моделируется как сеть с центральным обслуживанием, опускается так много деталей, что, например, можно представить только несколько типов изменений из приведенных в табл. 7.6. Для модели с тремя обслуживающими приборами можно учесть только те настроечные изменения, которые удается выразить изменениями m, $\bar{t}_{поиск}$, $\bar{L}$, $\bar{n}_{цп}$, $\bar{r}_{бр}$, $\bar{n}_{бр}$, $\bar{r}_{дс}$, и $\bar{n}_{дс}$ (см. табл. 7.2). Кроме того, как показано в примере 7.10, легко выйти за границы области пригодности данной модели, по крайней мере в простой ее версии, рассматриваемой на протяжении этой главы. Тем не менее мы будем анализировать эту проблему, используя данную простую модель для иллюстрации подхода, который может оказаться плодотворным, особенно если применить его к более сложным и точным моделям.

Наше первоначальное наблюдение состоит в том, что, согласно (4.101), $G(N)$ является симметрической функцией аргументов y_2 и y_3. Иначе говоря, их перестановка не влияет на значение $G(N)$. Таким образом, вследствие (4.106) ρ_1 также является симметрической функцией y_2 и y_3. Это заключение согласуется с наблюдением, что ρ_1 не может зависеть от нумерации нецентральных обслуживающих приборов в сети. Следовательно, кривые, соответствующие фиксированным значениям ρ_1 на плоскости (y_2, y_3) симметричны относительно биссектрисы первого квадранта. Если мы начертим их для заданного значения N, то получим семейство кривых, подобное приведенному на рис. 7.16. Кривизна этих линий меньше кривизны окружности для больших значений ρ_1, т. е. вблизи начала координат, и растет с уменьшением ρ_1. Заметим также, что с ростом N переход между двумя типами кривизны (т. е. меньшей и большей, чем кривизна окружности) происходит на меньшем расстоянии от начала координат. Чтобы получить аналитическое выражение для емкости системы, можно сделать различные приближения. Мы выберем такое, которое заменяет кривые для постоянных ρ_1 на рис. 7.16 дугами окружностей с центрами в начале координат. Дуга радиуса r будет соответствовать значению

$$\rho_1=\frac{1+2x+3x^2+\ldots+Nx^{N-1}}{1+2x+3x^2+\ldots+(N+1)\,x^N}, \qquad (7.41)$$

где $x=r/\sqrt{2}$. Заметим, что (7.41) получено из (4.106) и (4.101) подстановкой $s=3$, $y_2=y_3=x$. Ошибка этого приближения равна нулю на биссектрисе квадранта и максимальна на осях. Можно показать, что максимальная ошибка возрастает с увеличением N и (в процентном исчислении) относительно радиуса. Например, при $r=1$ эта ошибка

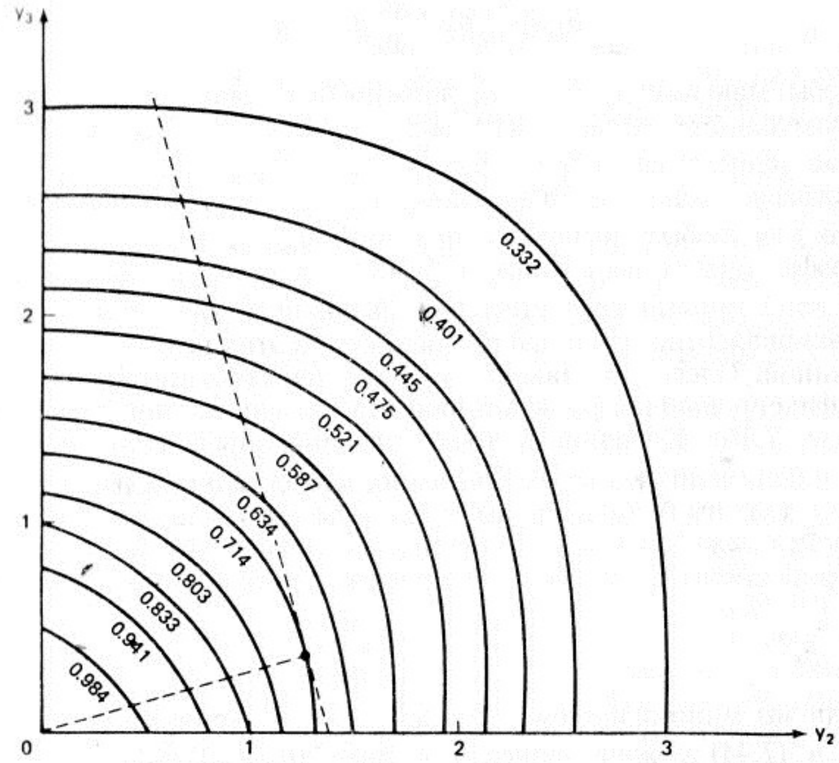


Рис. 7.16. Линии уровня ρ_1 для любой модели центрального обслуживания с тремя обслуживающими приборами при $N=5$.

составляет 3% при $N=5$ и 6% при $N=10$. Если $r=2$, ошибка равна 11% для $N=5$ и 25% для $N=10$. Ошибка 25% уже достаточно велика. Однако, если потребуется более высокая точность, можно будет использовать лучшие приближения. Кроме того, мы хотим оценить верхнюю границу для ρ_1, а рассматриваемая здесь аппроксимация имеет то преимущество, что дает значения ρ_1, больше действительных, если система работает в области, где кривизна большие кривизны окружности. Наконец, в практически интересных случаях система с оптимальной производительностью не работает слишком далеко от биссектрисы квадранта или от начала координат на плоскости (y_2, y_3) и, следовательно, следует ожидать, что ошибки окажутся всегда существенно меньше, чем приведенные выше максимальные значения.

Согласно нашей аппроксимации, кривые для фиксированных ρ_1 на плоскости (y_2, y_3) подчиняются уравнению

$$y_2^2+y_3^2=r^2. \qquad (7.42)$$

Координаты точки, соответствующей работе этой системы, на той же плоскости являются, согласно (4.97), функциями параметров модели (величин μ и b). Единственное ограничение, которому должны удовлетворять эти координаты, выражается соотношением $b_{11}+b_{12}+b_{13}=1$, которое, согласно уравнению (4.97), переходит в

$$b_{11}+\frac{\mu_2}{\mu_1}y_2+\frac{\mu_3}{\mu_1}y_3=1. \qquad (7.43)$$

Чтобы максимизировать $\overline{T}$, должно быть максимизировано N, а следовательно, m должно быть минимизировано. Если меняются остальные изменяемые параметры $\bar{t}_{\text{поиск}}$, $\bar{L}$, $\bar{n}_{\text{цп}}$, $\bar{r}_{\text{бр}}$, $\bar{n}_{\text{бр}}$, $\bar{r}_{\text{дс}}$ и $\bar{n}_{\text{дс}}$, это оказывает влияние на все коэффициенты в (7.43) (см. табл. 7.3). Однако для любых заданных значений $\bar{t}_{\text{поиск}}$, $\bar{L}$, $\bar{n}_{\text{цп}}$, $\bar{r}_{\text{бр}}$ и $\bar{r}_{\text{дс}}$ эти коэффициенты постоянны, а (7.43) является уравнением прямой. Точки этой прямой соответствуют различным значениям $\bar{n}_{\text{бр}}$ и $\bar{n}_{\text{дс}}$ (или, эквивалентно, b_{12} и b_{13}). Однако сумма этих величин должна быть постоянной. Очевидно, максимум ρ_1 должен соответствовать той кривой с фиксированным ρ_1, к которой эта прямая является касательной (см. рис. 7.16). Координаты точки касания определяют значения b_{12} и b_{13}, максимизирующие $\overline{T}$. Очевидно, что остальные изменяемые параметры должны быть выбраны так, чтобы эта точка оказалась по возможности ближе к началу координат. Из элементарной аналитической геометрии следует, что (7.43) является касательной к (7.42) при

$$r=\frac{(1-b_{11})\,\mu_1}{\sqrt{\mu_2^2+\mu_3^2}}. \qquad (7.44)$$

Если мы минимизируем r, мы достигаем максимума ρ_1. Числитель дроби в (7.44) можно записать в виде $(\bar{n}_{\text{цп}}-1)\,\bar{v}_{\text{цп}}/\bar{L}$. Поскольку $\bar{v}_{\text{цп}}/\bar{L}$ входит в выражение для $\overline{T}$ [см. (7.15)], мы не можем сделать вывод, что $\bar{L}$ нужно максимизировать. Напротив, можно показать, что ρ_1 убывает медленнее, чем $1/r$, при увеличении r. Таким образом, $\bar{L}$ необходимо минимизировать. Из (7.44) и уравнения для μ_3 в табл. 7.3, мы заключаем, что $\bar{t}_{\text{поиск}}$ следует минимизировать (очевидный вывод). Чтобы максимизировать μ_2 и μ_3, нужно минимизировать $\bar{r}_{\text{бр}}$ и $\bar{r}_{\text{дс}}$. Однако, если мы постулируем, что объем информации $\bar{n}_{\text{бр}}\bar{r}_{\text{бр}}+\bar{n}_{\text{дс}}\bar{r}_{\text{дс}}$, передаваемый каждым заданием из основной памяти во внешнюю и наоборот, должен оставаться постоянным, то уменьшение $\bar{r}_{\text{бр}}$ и $\bar{r}_{\text{дс}}$ будет приводить к увеличению $\bar{n}_{\text{бр}}$ и $\bar{n}_{\text{дс}}$. Следовательно, $\bar{n}_{\text{цп}}$ и числитель дроби (7.44) будут возрастать. Существуют оптимальные значения этих параметров, которые минимизируют (7.44) при ограничении, что объем данных, передаваемых каждым заданием, постоянен, а результаты соответствуют координатам точки касания. Отыскать эти оптимальные значения не просто. Если мы предположим, что μ_2 и μ_3 ограничены сверху временами доступа и что осуществимые изменения $\bar{r}_{\text{бр}}$ и $\bar{r}_{\text{дс}}$ не могут оказывать сильного влияния на соответствующие интенсивности обслуживания, то мы можем сохранить первоначальные значения $\bar{r}_{\text{бр}}$ и $\bar{r}_{\text{дс}}$. Чтобы упростить задачу, предположим, что $\bar{r}_{\text{бр}}=\bar{r}_{\text{дс}}$, как в табл. 7.2. Таким образом, значение $\bar{n}_{\text{цп}}$ можно получить, зная общий объем информации, передаваемой заданием, а $\bar{n}_{\text{бр}}$ и $\bar{n}_{\text{дс}}$ — из координат точки касания.

В качестве примера вычислим приблизительное значение $\overline{T}_{\max}$ для системы, модель которой характеризуется значениями параметров, приведенными в табл. 7.3, за исключением μ_3 (который равен 16.66 $с^{-1}$) и b_{12} и b_{13}, которые являются изменяемыми параметрами (это означает, что мы ограничиваемся здесь изменениями в размещении информации). Из (7.44) получим $r=1.22$. Уравнение (7.41) дает $\rho_1=0.781$, а (7.1) дает $\overline{T}_{\max}=3.25$ заданий/с. Мы также получим $b_{12}=0.88$ и $b_{13}=0.07$. На рис. 7.7 $\overline{T}_{\max}$ для этой же модели равно приблизительно 3.14 задания/с. Следовательно, ошибка приблизительно составляет 3%. Оптимальные значения b_{12} и b_{13} приблизительно равны соответственно 0.85 и 0.1.

7.4.2. Процедуры самонастройки

Большинство методов настроечной терапии, перечисленных в табл. 7.6, относится к характеристикам системы, для которых при заданной конфигурации и рабочей нагрузке можно определить оптимальный выбор. Как мы уже видели в разд. 7.4.1, это относится и к присоединению устройств к каналам, и к размещению файлов. Это в особенности справедливо для некоторых параметров операционной системы, таких, как квант времени ЦП в системах с разделением времени и степень мультипрограммирования в системах с виртуальной памятью [см. кривую (с) на рис. 7.10]. Другим примером является выбор одного алгоритма из многих алгоритмов управления ресурсами. Если производительность определяется единственным индексом или критерием, то каждый раз, когда выбор одного из нескольких известных алгоритмов оказывает некоторое влияние на этот индекс, один из этих алгоритмов дает наилучшую производительность. Однако оптимальный выбор таких характеристик системы меняется с рабочей нагрузкой. Следовательно, как только рабочая нагрузка меняется, система «расстраивается» и требует повторной настройки. Как уже упоминалось несколько раз в предыдущих главах, спектральный анализ параметров рабочей нагрузки, таких, как запросы на ресурсы, показывает, что их спектр частот охватывает несколько порядков величины. Диапазон периодов простирается от миллисекунд до месяцев и даже лет. Методы настроечной терапии, которые можно применить при периодах порядка месяцев, не могут учитывать более высокочастотные изменения и должны использовать средние значения параметров рабочей нагрузки за периоды того же порядка. Привязку уст-

ройств к каналам нельзя менять чаще, чем, скажем, самое большее несколько раз в году. Следовательно, этот метод терапии можно применять для перенастройки системы в том и только в том случае, когда соответствующие характеристики месячной или квартальной рабочей нагрузки значительно отклонились от соответствующих показателей, которые существовали в момент предыдущей настройки.

Однако можно применять чаще, например, методы терапии, состоящие в подборе значений параметров операционной системы или в переходе от одного алгоритма управления ресурсами к другому. Если бы удалось автоматизировать процесс настройки, эти методы позволили бы нам подстраивать систему к изменениям рабочей нагрузки, обладающим очень короткими периодами. Автоматизация процесса настройки существенна, если система должна реагировать на высокочастотные изменения своей рабочей нагрузки. Можно представить механизм автоматической настройки, действующий в гораздо большем масштабе времени, который, например, перемещает файлы с одного устройства на другое для улучшения производительности системы. В принципе автоматическая настройка может производиться извне обычно с помощью аппаратного измерителя с запоминаемой программой (см. разд. 2.3.4 и 2.5). Однако наибольшей популярностью до сих пор пользуется подход, использующий для этой цели саму систему. Мы будем называть такую систему *самонастраивающейся*.

Этап диагностики в любом процессе автоматизированной настройки состоит из измерения соответствующих величин и интерпретации результатов. Это требует процедуры непрерывного наблюдения и простого и быстрого алгоритма принятия решений, определяющего новые значения параметров операционной системы или новые стратегии управления ресурсами. За какими величинами нужно вести непрерывное наблюдение? Как показано на рис. 7.17(a) и (b), мы можем измерять параметры рабочей нагрузки *(прямое управление)* или индексы производительности *(обратная связь)*. Вообще говоря, первый подход был бы проще, если известны параметры рабочей нагрузки, влияющие на производительность, а значения этих параметров доступны. Это было бы и более эффективно, потому что система могла бы подготовиться к поступающей рабочей нагрузке. Это редко удается, однако, различные процедуры самонастройки заимствуют у этого подхода идею использования производительностей отдельных заданий вместо производительности всей системы для принятия своих решений. Так как эти производительности заранее неизвестны, их приходится получать в реальном времени. Таким образом, эти процедуры являются *гибридными*, использующими как прямое управление, так и обратную связь. Их можно представить схематически так, как показано на рис. 7.17(c).

Любое измерение, производимое в схемах с обратной связью или в гибридных схемах, должно удовлетворять важному требованию: вероятность того, что измеряемая величина остается относительно стабильной на протяжении периода от начала выполнения решения о настройке до ближайшего момента, когда можно сделать новое изменение, должна быть велика. Для управления с обратной связью характерно то, что решение о регулировке основывается на измерениях эффектов, причины которых могут исчезнуть к моменту, когда станут

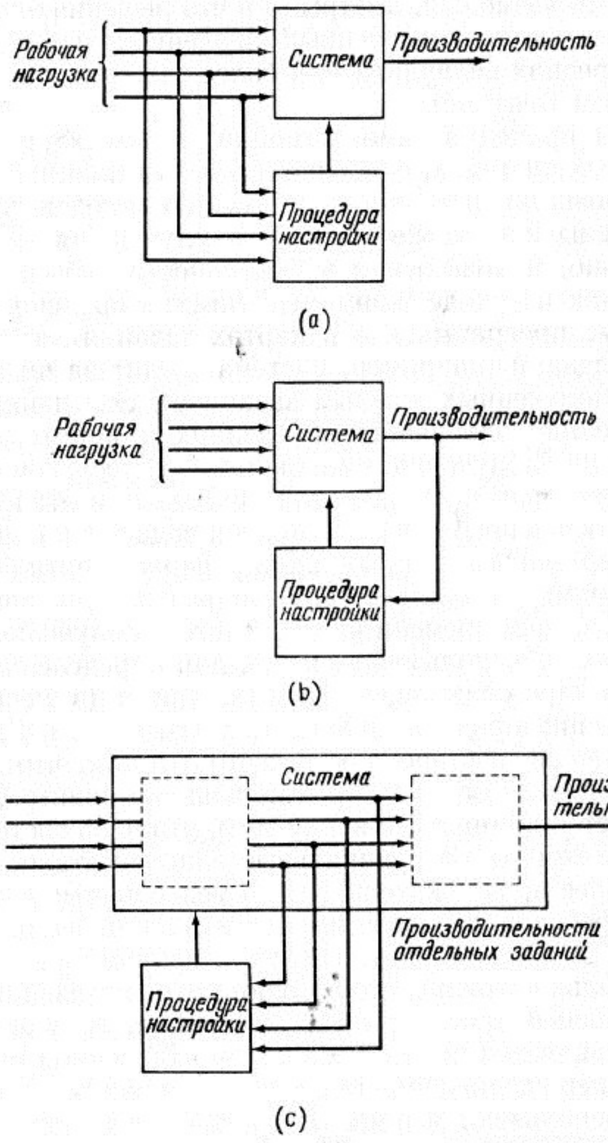


Рис. 7.17. Схемы автоматической настройки: (a) — прямое управление, (b) — управление с обратной связью, (c) — гибридное управление.

ощутимыми последствия принятого решения. Если это верно, производительность системы может существенно уменьшиться и даже может возникнуть нестабильность. Приведенное выше условие требует, чтобы измеренные данные, использующиеся для выбора настроечных изменений, представляли производительность системы или отдельных зада-

ний не только в тот момент, когда они были собраны, но также с высокой вероятностью на протяжении интервала, в котором будут действовать эти изменения. Непосредственные следствия этого требования состоят в том, что этапы диагностики и терапии должны быть, насколько это возможно, быстрыми и что решения о настройке должны основываться не на мгновенных значениях измеренных величин, а скорее на средних величинах или более сложных прогнозах поведения в ближайшем будущем.

Примеры процедур самонастройки можно обнаружить во многих операционных системах, особенно в области планирования. Практически планировщики применяют множество методов для ускорения реакции системы и в то же время для улучшения ее балансировки, а следовательно, и повышения эффективности. Обычно измерения производятся как на входе (например, лимиты времени и размеры памяти, заданные программистом в картах задания), так и во время выполнения заданий (например, система следит за числом квантов времени, уже полученных каждым заданием). Эти данные используются для назначения подходящих внутренних приоритетов заданиям так, чтобы порядок получения ими системных ресурсов был в некотором смысле оптимальным. В пакетной системе лимиты времени и памяти могут быть основанием для постановки задания в одну из нескольких входных очередей, из которых планировщик памяти берет задания для загрузки в память в соответствии с определенным алгоритмом. В системе с разделением времени число квантов, полученных определенным заданием, может использоваться как для определения срочности предоставления ему следующего кванта, так и (в некоторых случаях) для определения продолжительности этого кванта. Разумное объяснение последней предосторожности состоит в том, что задание, которое уже израсходовало определенное количество квантов, по всей вероятности, является длинным, и ввиду того, что цель системы — давать более быстрые ответы на тривиальные запросы, такое задание должно получать более продолжительные, но менее частые кванты. В действительности требования разработчиков к планировщику настолько противоречивы, а его взаимоотношения с другими распорядителями ресурсов настолько сложны, что абсолютная оптимизация производительности с помощью процедуры самонастройки, встроенной в планировщик, представляется пока что слишком отдаленной целью. Более подходящим утверждением является то, что различные процедуры самонастройки используются и часто успешно в планировщиках для улучшения производительности системы или некоторых аспектов производительности. Например, важным аспектом, с которым планировщик постоянно имеет дело, является упоминавшееся выше требование обеспечения малого времени ответа или прохождения задания для очень коротких заданий.

Настройка загрузки является примером процедуры самонастройки, которую можно найти в диалоговых системах [см. Уилкс (1971)]. В системе CTSS, например, использована схема с обратной связью,

подобная представленной на рис. 7.17(b). Наблюдаемыми переменными были средняя длина всех очередей и флажок переполнения быстрого барабана, который устанавливался, когда этот барабан оказывался заполненным и надо было произвести копирование заданий пользователей на гораздо более медленный диск [см. дополнение к работе Уилкса (1971), написанное Миллсом]. Управляемой переменной было количество пользователей системы. Если это число превосходило максимально допустимое в условиях перегрузки, диалог с пользователем автоматически прекращался.

Теперь мы обсудим вкратце другую важную проблему, для которой могут оказаться полезными процедуры самонастройки: управление степенью мультипрограммирования в системе с виртуальной памятью. Цель этого, как можно сразу же убедиться, взглянув на кривую (c) рис. 7.10, состоит в том, чтобы сохранить рабочую точку вблизи максимума $\overline{T}$ и избежать толкотни. Поддержание N постоянным и равным значению, максимизирующему $\overline{T}$, решило бы проблему, если бы кривая $\overline{T}(N)$ не зависела от времени. Однако, как видно из рис. 7.12, флуктуации интенсивностей страничных отказов для отдельных заданий заставляют кривую двигаться, а если N сохраняется постоянным, система вполне может войти в режим толкотни, когда одно или более заданий вызывают такое количество страничных отказов, что это парализует страничное устройство внешней памяти (устройство «подкачки»). Мы уже видели в разд. 7.4.1, что переменное распределение памяти позволяет надеяться, что ситуация облегчится. Это можно показать на примере.

Предположим, что в системе, которую мы анализировали в примере 7.3, поведение всех заданий (в отношении ссылок на память) одинаково и характеризуется двумя параметрами $a=6.25$ страниц$^{-k}$ и $k=2$. Мы уже видели в примере 7.3, что максимум $\overline{T}$ для этой системы с данной рабочей нагрузкой соответствует значению $N=3$. Кривая (c) рис. 7.10 показывает, что при этих условиях $T=2.58$ задания/с, а, согласно рис. 7.12, соответствующее среднее значение общей интенсивности страничных отказов равно приблизительно 169 страничным отказам/с. Теперь предположим, что поведение рабочей нагрузки неожиданно изменяется так, что это изменение можно описать уменьшением a со значения 6.25 до значения 2.5 для всех заданий. Если N по-прежнему равно 3, то, согласно (7.14), $\overline{f}$ возрастает до 421 страничного отказа/с, а $\overline{T}$ падает приблизительно до 1.5 заданий/с. Однако, если N уменьшается и становится равным 2, то $\overline{f}$ падает до 125 страничных отказов/с, а $\overline{T}$ уменьшается только до 2.08 задания/с. Таким образом, производительность уменьшается теперь не на 42%, а всего на 19%, и этот выигрыш происходит только за счет понижения степени мультипрограммирования. Этот результат легко проверить графически на кривых $\overline{T}$ рис. 7.12. Заметим, что схема самонастройки, которая следила бы за коэффициентом использования ЦП ρ_1 и пыталась поддер-

жать его на высоком уровне с помощью увеличения N при снижении ρ_1, немедленно привела бы к катастрофической толкотне. Читатель может увидеть на рис. 7.12, что произошло бы, вспомнив, что по (7.14) $\bar{f}$ пропорционально N^{k+1}.

С другой стороны, предположим, что поведение рабочей нагрузки улучшается в некоторый момент времени. Например, предположим, что a увеличивается в 2.5 раза по сравнению с первоначальным значением: $a=15.625$ страницы$^{-k}$. Если $N=3$, то $\bar{f}$ уменьшается до 67.5 страничного отказа/с, а $\overline{T}$ возрастает до 2.98 задания/с. Однако если N увеличить до 4, то мы получим $\bar{f}=160$ страничных отказов/с, а $\overline{T}=3.08$ задания/с. Таким образом, мы получаем улучшение приблизительно на 20% вместо 15% благодаря тому, что разрешаем расположить в памяти еще одно задание.

Интересно посмотреть, какова будет интенсивность страничных отказов каждого задания в различных только что описанных ситуациях. Первоначально $\bar{f}/N$ при $N=3$ приблизительно равно 56 страничным отказам/с. Когда a становится равным 2.5, $\bar{f}/N$ возрастет до 140, если N сохраняют постоянным, но окажется равным только 62, если сделать $N=2$. Когда a равно 15.625, $\bar{f}/N$ оказывается равным 22 для $N=3$ и 40 для $N=4$. Таким образом, процедура управления, поддерживающая отношение $\bar{f}/N$ постоянным (или по крайней мере уменьшающая отклонения $\bar{f}/N$), улучшает производительность по сравнению с неотрегулированной системой. Если поддерживаемое постоянным значение $\bar{f}/N$ выбрано правильно, эта процедура предотвращает толкотню. Каждый раз, когда $\bar{f}$, возрастая, выходит за определенную границу, N уменьшается настолько, чтобы значение f/N, которое, согласно (7.14), пропорционально N^k, оставалось приблизительно постоянным. Из рис. 7.12 ясно видно, что попытка поддерживать $\bar{f}$ постоянным была бы ошибочной. Заметим, что подход, использующий управление числом страничных отказов отдельных заданий, позволяет нам использовать гибридную схему, подобную представленной на рис. 7.17(c). Заметим также, что, как видно из примера 7.10, условие равенства интенсивностей отказов является оптимальным при переменном распределении памяти и постоянном N. И, разумеется, именно такую ситуацию мы имеем между любыми двумя последовательными моментами изменения N.

Как можно поддерживать постоянной интенсивность страничных отказов программы? Если функция времени жизни представляет поведение программы адекватно, эта цель может быть достигнута выделением программе пространства, равного

$$m(a, k) = \left(\frac{\bar{v}_{\text{цп}}}{a}\,\bar{e}^*\right)^{1/k}, \tag{7.45}$$

где $\bar{e}^*$ является величиной, обратной той интенсивности страничных отказов, которая выбрана для всех программ. Заметим, что в (7.45), как и в (7.38), m пропорционально $a^{-1/k}$. Так как a и k изменяются, можно вычислить m по (7.45), а степень мультипрограммирования по формуле $N=M/m$. На практике измерение параметров модели средних значений, таких, как a и k, очень затруднено. Особенно трудно удовлетворить требованиям, сформулированным несколькими абзацами выше. Мы можем, однако, попытаться предсказать значение m, которое потребуется в ближайшем будущем, чтобы сделать интенсивность страничных отказов каждой программы приблизительно равной желаемой величине ($1/\bar{e}^*$). Деннинг (1968a) предположил, что m для каждой программы можно оценить, полагая его равным размеру рабочего множества программы в момент оценки.

Рабочее множество программы $W(t, \tau)$ в момент t с *размером окна* τ есть набор страниц, на которые ссылалась программа в течение самого последнего интервала времени ЦП длительности τ. Иначе говоря, это набор страниц, на которые были ссылки в интервале от $t-\tau$ до t. Очень важно помнить, что речь идет об интервале времени ЦП, фактически использованного программой (иногда называемого также *виртуальным временем*), а не об интервале реального времени.

Стратегия рабочего множества является одним из наиболее удачных средств самонастройки, когда-либо использовавшихся в вычислительных системах. Пространство m, которое должно быть выделено программе в ближайшем будущем, оценивается в виртуальный момент времени t как

$$m(t) = w(t, \tau) = |W(t, \tau)|, \tag{7.46}$$

т. е. совпадает с размером рабочего множества программы в этот момент времени. Величиной N управляют так, чтобы она соответствовала оценке (7.46). Если сумма всех областей памяти, которые должны быть выделены программам, превосходит M, одна из программ переводится в состояние ожидания и разгружается из памяти. Наоборот, когда эта сумма настолько мала по сравнению с M, что по оценке m (т. е. рабочего множества) еще одна программа может быть размещена в основной памяти, такая программа активируется и загружается в память. Читателю следует обратить внимание на то, что измерение $W(t, \tau)$ используется не только для оценки m, но также и для принятия решения, какая часть каждой программы должна сохраняться в памяти, чтобы интенсивность страничных отказов программы не очень сильно отличалась от $1/\bar{e}^*$ в ближайшем будущем. Эта часть совпадает, по предположению Деннинга, с множеством страниц $W(t, \tau)$.

Очевидно, выбор τ является решающим. Если τ слишком велико, область памяти, отведенная каждой программе, окажется слишком большой, а N и $\overline{T}$ окажутся меньше, чем это было бы возможно и желательно. Если τ слишком мало, оценка потребности в памяти для каждой программы окажется заниженной, а N будет слишком велико. Таким образом, $\overline{T}$ понизится вследствие толкотни. В схеме с постоянным и одинаковым для всех размером окна значение τ должно выбирать-

ся с помощью итеративной процедуры ручной настройки и, возможно, периодически подстраиваться. Однако были предложены различные процедуры самонастройки, в которых τ меняется автоматически [см. Гендерсон и Родригес-Роселл (1974)] или в которых для различных программ используются различные значения τ (см. в работе Чу и Ондербека (1972) схему с обеими характеристиками) и даже различные τ для различных страниц [Прив (1973)]. Эти схемы используют эмпирические наблюдения, что различные программы проявляют различное поведение, и нацелены на то, чтобы улучшить эффективность приведенного выше метода прогнозирования m.

Крайне важно отметить, что стратегия рабочего множества, так же как и ее модификации, позволяет успешно оценивать пространство, необходимое для каждой программы, и, следовательно, эффективно управлять степенью мультипрограммирования, только если поведение программы в прошлом и настоящем может быть использовано для прогнозирования поведения в ближайшем будущем. Эмпирически было обнаружено, что для большинства программ это справедливо. Картина ссылок часто характеризуется феноменом *локальности*. Неформально это означает, что страницы, на которые были ссылки в ближайшем прошлом, обладают наибольшей вероятностью обращения к ним в непосредственном будущем по сравнению с остальными страницами программы. Очевидно, что очень локальная программа, в которой команды и ссылки на данные концентрируются на нескольких страницах в течение относительно длительного периода времени, весьма облегчает прогнозирование (например, методом рабочего множества), основанное на предыдущем поведении программы в течение некоторого предшествующего интервала виртуального времени. Свойство локальности и его наиболее интересные аспекты будут проанализированы более подробно в гл. 9.

Еще один алгоритм самонастройки для управления степенью мультипрограммирования в системе с виртуальной памятью был предложен Баделем и др. (1975). Этот алгоритм относится к типу алгоритмов с обратной связью [см. рис. 7.17 (b)]. Измеряемый индекс является линейной комбинацией коэффициентов использования различных ресурсов в течение интервала реального времени, прошедшего с момента последнего измерения. Заметим, что измерения производились при поступлениях и уходах заданий или периодически. Смещенная экспоненциальная оценка, основанная на значениях индекса, измеренных в прошлом, затем используется для принятия решения об увеличении, уменьшении или сохранении N.

7.4.3. Методы развития

Изменения установки, состоящие в замене или добавлении компонентов оборудования, были названы в разд. 7.2 *методами развития*. Термин *развитие* часто используется для обозначения замены системы целиком на более мощную, обычно сделанную тем же изготовителем

и часто принадлежащую тому же семейству или той же серии компьютеров. В соответствии с нашими определениями, мы рассматриваем это как приобретение ресурса и, следовательно, проблему выбора, тогда как под развитием мы понимаем относительно малые изменения конфигурации оборудования данной системы. При этих изменениях возможно исключение некоторых компонентов оборудования (в этом случае соотношение стоимость — производительность улучшается за счет снижения стоимости, а не за счет повышения производительности). Разумеется, обычное утверждение о нечеткости большинства наших определений и их ограниченности должно быть повторено и здесь.

Когда администратору установки следует рассматривать развитие вместо настройки? Заключение, к которому мы пришли в разд. 7.4.1 относительно этой проблемы, состоит в том, что невозможно установить общие правила. Очевидно, существуют случаи, когда настройка не может дать никаких существенных улучшений. Систему, ресурсы которой одинаково насыщены или близки к насыщению, следует развивать или заменить на более мощную, если только ее рабочая нагрузка не может быть резко уменьшена. С другой стороны, как было сказано в разд. 7.4.1, некоторые методы настройки могут оказаться настолько же дорогими, как и некоторые изменения в оборудовании, и даже дороже. Если бы были возможны оба метода терапии, решение следовало бы принимать по критерию цена — выигрыш. Следует также принимать во внимание будущие нужды коллектива пользователей. Разумеется, развитие не исключает, а в действительности даже часто требует настройки (см. задачу 7.14).

Теперь мы обсудим простой пример применения критерия цена — выигрыш в контексте работы по развитию. В разд. 7.3.2 мы определили формально понятие узкого места через частные производные индекса производительности по интенсивностям обслуживания различных ресурсов. Ресурс, соответствующий максимальной производной, был назван создающим узкое место в системе. Важно понять, что не всегда лучшим способом повышения производительности системы будет устранение узкого места. Узкое место определенно нужно устранять при сильной несбалансированности системы. Однако, если несбалансированность минимальна, может оказаться удобнее воздействовать на компонент, не создающий узкого места.

Пусть c_i — стоимость i-го ресурса (оборудования), т. е. количество вложенных в него денег. Прирост интенсивности обслуживания μ_i можно, вообще говоря, получить, увеличив его стоимость на Δc_i. Обе переменные μ_i и c_i обычно принимают дискретные значения, а функция $\mu_i(c_i)$ определена только в нескольких точках, соответствующих моделям этого компонента, имеющимся на рынке в данный момент времени. Отметим также, что вследствие технического прогресса $\mu_i(c_i)$ зависит от времени. Даже если игнорировать эти факты, относящиеся к функции $\mu_i(c_i)$, вполне возможно, что в средне-несбалансированной системе, в которой j-й ресурс создает узкое место, можно

найти другой ресурс (i-й), такой, что

$$\frac{\partial \bar{T}}{\partial c_i} > \frac{\partial \bar{T}}{\partial c_j}, \tag{7.47}$$

В этом случае больший прирост производительности можно получить, вкладывая определенное количество денег в развитие i-го ресурса, чем вкладывая то же количество денег в развитие j-го ресурса. Заметим, что из (7.47) и утверждения, что j-й ресурс создает узкое место, следует

$$\frac{\partial \mu_i}{\partial c_i} > \frac{\partial \mu_j}{\partial c_j}. \tag{7.48}$$

Основываясь на (7.47), мы можем сформулировать *определение узкого места, ориентированное на стоимость и производительность*: говорят, что i-й ресурс создает узкое место (в смысле стоимости и производительности), если (7.47) справедливо для всех $j \neq i$. Это определение можно распространить на все типы ресурсов, включая те, производительность которых не характеризуется скоростью обслуживания. Например, оно применимо к размеру основной памяти и может использоваться для решения вопроса, следует ли, скажем, приобретать более быстрый диск или дополнительный блок памяти. Для всех практических целей частные производные в приведенных выше соотношениях, вследствие трудности их вычисления и вследствие дискретной природы большинства входящих в них переменных, будут заменены соответствующими отношениями приращений.

Пример 7.11. Настроенную версию ПРОП-системы пакетной обработки, описанную в нескольких предыдущих примерах, предстоит развить с целью увеличения ее средней пропускной способности $\bar{T}$ по меньшей мере на 10% от старого значения 3.13 до по крайней мере 3.44 задания/с. Система как раз такова, что соответствует максимуму $\bar{T}$ на рис. 7.7. Значения ее параметров совпадают с перечисленными в табл. 7.3, кроме μ_3, равного 16.66 $с^{-1}$, и b_{12} и b_{13}, равных соответственно 0.85 и 0.1. В разд. 7.3.2 мы уже видели, что, будучи самым быстрым из нецентральных обслуживающих приборов, барабан создает узкое место в данной ситуации максимального $\bar{T}$. Однако было бы, вообще говоря, ошибкой пренебречь остальными возможностями развития. В табл. 7.8 приведены результаты гипотетического изучения рынка и наши вычисления, основанные на модели рис. 7.5. Возможности, включенные в табл. 7.8, являются некоторыми из наилучших доступных, выбранных с учетом дополнительных ограничений на требуемый размер памяти и совместимости с остальной частью конфигурации.

Наибольшее увеличение средней пропускной способности на доллар затрат достигается при увеличении памяти. Если допустимо ухудшение среднего времени прохождения $\bar{t}$ на 1 с, то добавление 128K слов основной памяти является самым дешевым путем достижения желаемой производительности. Наибольшее улучшение $\bar{T}$ было бы получено развитием барабана, который создает узкое место. Однако замена барабана может оказаться весьма дорогой, как и замена центрального процессора. Разумеется, легко представить ситуации, в которых развитие барабана или даже ЦП было бы удобнее расширения памяти. Веские доводы в пользу этого могли бы включать нежелательность увеличения времени прохождения и большее удобство такого изменения, рассматриваемого как начальный шаг гораздо более далеко идущего плана развития. Заметим также, что табл. 7.8 не показывает потенциальных возможностей различных решений относительно дальнейшего увеличения $\bar{T}$ в про-

цессе настройки. Эти потенциальные возможности вполне могут входить в число аргументов при принятии решения о развитии системы. В задаче 7.16 читателя просят оценить их, используя приближенный метод, введенный в конце разд. 7.4.1. ■

Таблица 7.8

Соотношение стоимость — производительность для различных методов развития, изученных в примере 7.11

Развиваемый компонент	Степень развития	Цена развития (Δc) [долл.]	Предсказанное улучшение ($\Delta \bar{T}$) [заданий/с]	Улучшение на единицу цены ($\Delta \bar{T}/\Delta c$) [10^3 заданий/долл.с]	Предсказанное изменение $\bar{t}$ [с]
ЦП	$\Delta\mu_1 = 53.3\ с^{-1}$	85000	0.38	0.0044	—0.17
Барабан	$\Delta\mu_2 = 15.3\ с^{-1}$	60000	0.35	0.0058	—0.16
Диск	$\Delta\mu_3 = 35.6\ с^{-1}$	35000	0.09	0.0025	—0.04
Основная память	$\Delta M = 32K$ слов	10000	0.12	0.012	+0.24
	$\Delta M = 64K$ слов	20000	0.20	0.010	+0.50
	$\Delta M = 96K$ слов	30000	0.27	0.009	+0.75
	$\Delta M = 128K$ слов	40000	0.32	0.008	+1.01

Методы, которыми можно воспользоваться для прогнозирования влияния развития системы на производительность, относятся к тому же типу, что и методы оценки результатов настроечных изменений. Для этой цели можно использовать как методы моделирования, так и аналитические. Поскольку ошибки в решениях о развитии обычно обходятся гораздо дороже и их труднее исправить, чем в случае настройки системы, возможность использовать надежные и относительно простые методы прогнозирования даже важнее для методов развития, чем для настройки. Мы теперь распространим обсуждение примеров 7.8 и 7.9 на некоторые случаи терапии развития.

Пример 7.12. Систему, рассмотренную в примерах 7.6 и 7.8 (см. табл. 7.5), предстоит усовершенствовать путем подходящего изменения конфигурации ее оборудования. В примере 7.6 наш диагноз, основанный на профиле использования системы рис. 7.15 (а), констатировал большое время простоя системы, несбалансированность каналов и недостаточную величину перекрытий — три основные приблемы производительности этой системы. В примере 7.8 мы изучили последние две проблемы и, в частности, несбалансированность каналов и результаты принятия некоторых мер ее исправления. На мысль об изменении, не увеличивающем $\bar{T}$, но возможно удобном с точки зрения критерия «стоимость — производительность», наводил крайне низкий коэффициент использования канала 2. Насколько бы упала производительность, если бы мы исключили канал 2, а все устройства присоединили бы к каналу 1? Для решения этой задачи полезно уравнение времени системы вида

$$t_{\text{общ}} = t_{\text{только ЦП}} + t_{\text{любой канал занят}} + t_{\text{система бездействует}}, \tag{7.49}$$

при $t_{\text{любой канал занят}} = (\rho_2 + \rho_3 - \rho_{23}) t_{\text{общ}}$.

Предположим, что исключение канала 2 влияет только на $t_{\text{любой канал занят}}$, которое становится равным

$$t'_{\text{любой канал занят}} = (\rho_2 + \rho_3)\, t_{\text{общ}}. \tag{7.50}$$

Таким образом,

$$\frac{t'_{\text{общ}}-t_{\text{общ}}}{t_{\text{общ}}}=\rho_{23}=0.41\%, \tag{7.51}$$

очень малое снижение $\overline{T}$ вполне соответствует связанной с этим экономии.

Как объяснялось в примере 7.4, большое время простоя системы может быть результатом нехватки заданий, медлительности операторов или неперекрывающихся операций поиска на диске. Возможным методом развития явилась бы замена дисков на диски с более высокой скоростью поиска. Уравнение времени системы позволяет нам приблизительно предсказать результат такой терапии. Однако мы уже измерили долю общего времени, в течение которого единственным действием системы является поиск на диске. Таким образом, $t_{\text{система бездействует}}$ должно быть разделено на сумму двух частей: $t_{\text{только поиск}}$ и $t_{\text{только ожидание}}$. Уравнением времени системы является

$$t_{\text{общ}}=t_{\text{только ЦП}}+t_{\text{любой канал занят}}+t_{\text{только поиск}}+t_{\text{только ожидание}}, \tag{7.52}$$

а новое уравнение будет

$$t'_{\text{общ}}=t_{\text{только ЦП}}+t_{\text{любой канал занят}}+\alpha t_{\text{только поиск}}+t_{\text{только ожидание}}, \tag{7.53}$$

где α — отношение среднего времени поиска для новых дисков к той же величине для старых. Следовательно,

$$\frac{t_{\text{общ}}-t'_{\text{общ}}}{t_{\text{общ}}}=(1-\alpha)\frac{t_{\text{только поиск}}}{t_{\text{общ}}}. \tag{7.54}$$

Если $t_{\text{только поиск}}/t_{\text{общ}}=28\%$, а $\alpha=0.7$, то улучшение, даваемое (7.54), равно 8.4%. Заметим, что если к этим каналам были бы подсоединены устройства иного типа, чем диски (т. е. лентопротяжные устройства), то следовало бы из времени простоя других компонентов системы выделить время работы *только лентопротяжных устройств*, а множитель α отнести к *чистому времени поиска на диске*. ■

Пример 7.13. Система UNIVAC 1108, описанная в примере 7.9, измерялась под управлением набора эталонов, подготовленного Хьюзом и Мо (1973). Результирующие коэффициенты использования приведены в столбце ρ_i табл. 7.9. Читатель может видеть, что все они довольно малы. Такое недостаточное использование компонентов системы может быть следствием различных причин. Если большая часть его может быть объяснена временем простоя системы, то причину, вероятно, следует искать

Таблица 7.9

Оценка результатов развития: расширение памяти (см. пример 7.13)

Номер обслуживающего прибора, i	Ресурс	Коэффициент использования до изменения ($M=128K$ слов), ρ_i	Оценка коэффициента использования после изменения ($M=192K$ слов), $\hat{\rho}'_i$	Измеренный коэффициент использования после изменения, ρ'_i
1	ЦП	0.59	0.79	0.77
2	Барабан FH432	0.26	0.35	0.39
3	Барабан FH880	0.26	0.35	0.38
4	Барабан FASTRAND	0.56	0.75	0.74
5	Магнитная лента	0.25	0.33	0.35

Источник: Хьюз и Мо (1973).

среди обсуждавшихся в примере 7.4. Однако нехватку заданий и медлительность операторов следует исключить из числа возможных причин из-за управляемой среды, в которой проводился эксперимент. Время чистого поиска в этой системе не существует, но его можно заменить временем чистого поиска на ленте. Другой гипотезой, которая могла бы объяснить недостаточное использование всех ресурсов, является гипотеза, идентифицирующая узкое место с памятью. Другими словами, возможно, что вследствие ограничений размера памяти степень мультипрограммирования недостаточно высока, чтобы поддерживать приемлемую занятость компонентов системы.

Проверку этой гипотезы и оценку улучшения, которое достижимо при расширении основной памяти, можно выполнить с помощью подхода, проведенного в примере 7.9. Хьюз и Мо (1973) использовали модель центрального обслуживания для получения кривой $\rho_1(N)$. Значения y_2, y_3, y_4 и y_5, необходимые для вычисления ρ_1 при каждом N, можно вычислить по измеренным значениям коэффициентов использования ρ_i (см. (7.32)). С помощью кривой $\rho_1(N)$ можно определить N интерполяцией, как в примере 7.9. В данном случае мы получим $\overline{N}\approx 2.5$. Объем памяти, который можно выделить заданиям пользователей, равен $M\approx 64K$ слов. Остальная часть памяти, состоящей из $128K$ слов, занята операционной системой EXEC 8. Заметим, что измеренные коэффициенты использования включают также накладные расходы операционной системы, тогда как M — память, которая может быть распределена между заданиями пользователей. Однако предположения, сделанные нами для использования модели центрального обслуживания, настолько грубы, что следует либо принять это расхождение, либо ввести коррекцию измеренных коэффициентов использования для учета накладных расходов.

Если M будет удвоено, то ожидается, что $\overline{N}=M/\overline{m}$ также должно удвоиться. Таким образом, $\overline{N}'=5$ при $M'=128K$ слов. Значение $\hat{\rho}'_1$ можно определить по кривой $\rho_1(N)$, предположив, что расширение памяти не влияет ни на вероятности запросов p_{ij}, ни на интенсивности обслуживания различных компонентов. По $\hat{\rho}'_1$ и значениям y_2, y_3, y_4 и y_5 мы можем оценить $\hat{\rho}'_2$, $\hat{\rho}'_3$, $\hat{\rho}'_4$ и $\hat{\rho}'_5$ (см. табл. 7.9). Измеренные коэффициенты использования после расширения памяти под управлением того же набора эталонов показали удивительно хорошее согласие с предсказанными значениями, что видно из табл. 7.9.

Эти результаты также подтверждают нашу гипотезу: удвоение размера памяти увеличивает $\overline{T}$ (более чем на 30%). Заметим, что модель центрального обслуживания никогда не простаивает. Следовательно, ее нельзя использовать для оценки результатов тех изменений, которые уменьшают время простоя системы. ■

7.5. МЕТОДЫ ОЦЕНКИ В РАБОТАХ ПО УСОВЕРШЕНСТВОВАНИЮ

В предыдущих разделах этой главы много внимания было уделено методам оценки, используемым в работах по усовершенствованию. В этом разделе мы подведем краткие итоги наиболее важным соображениям, высказанным по данному предмету.

Анализ производительности играет существенную роль в методиках усовершенствования. Анализ производительности измеренной версии улучшаемой системы необходим как при формулировке гипотезы на этапе диагностики, так и при прогнозировании результатов предложенной терапии.

Поскольку улучшаемая система по определению существует, то к ней можно применить методы измерения. Однако только очень редко эти методы оказываются удобными для анализа результатов изменений, так как производить изменения в оборудовании и в программном обеспечении довольно дорого и рискованно. Измерение полезно для сбора

симптомов посредством непрерывного наблюдения или специального исследования (например, когда необходимы новые доказательства выдвинутой гипотезы). Оно часто не подходит для целей прогнозирования, хотя можно представить себе некоторые ограниченные применения методов измерения в этой области. Некоторые изменения системы можно, например, имитировать подходящими изменениями рабочей нагрузки, которые вводятся достаточно просто, если система управляется исполняемой моделью искусственной рабочей нагрузки, в особенности если это набор синтетических заданий. Еще одно применение методов измерения в этой области основано на экстраполяции. Например, результат добавления одного модуля памяти можно приблизительно оценить путем исключения одного из модулей памяти из системы и сравнивая полученную производительность с исходной. Из-за нелинейностей, присущих поведению вычислительных систем, результаты таких экстраполяций следует использовать с большой осторожностью.

Прогнозирование результатов изменений системы приходится, следовательно, делать в большинстве случаев с помощью имитационного моделирования или аналитических методов. Несколько примеров использования таких методов, в первую очередь аналитических, подробно обсуждалось в разд. 7.3 и 7.4. Значения параметров этих моделей и их калибровка основывались на результатах измерения неизменной системы. Как мы видели в разд. 6.4, относительная точность модели в общем случае важнее ее абсолютной точности. Другими словами, модель, позволяющая оценить, скажем, индекс $\overline{T}$ с точностью 20%, а отклонение $\Delta\overline{T}$ с максимальной ошибкой в 1%, может оказаться предпочтительнее, чем модель, у которой доверительные интервалы для $\overline{T}$ и $\Delta\overline{T}$ составляют 5%. Однако очень часто важное значение имеет лишь порядок величины $\Delta\overline{T}$ и равномерная точность прогноза $\Delta\overline{T}$ при различных изменениях. Соображения, приведенные в разд. 6.3.1 и 6.4 относительно сравнения производительностей и его влияния на выбор подходящих методов оценки, можно повторить для исследований по усовершенствованию. Однако в этих исследованиях большинство проблем сильно упрощается (по сравнению с задачами выбора) тем, что сравниваемые системы весьма похожи друг на друга.

7.6. ОПИСАНИЕ РАБОЧЕЙ НАГРУЗКИ В РАБОТАХ ПО УСОВЕРШЕНСТВОВАНИЮ

Замечание, что работы по усовершенствованию требуют сравнения производительностей и что это сравнение ставит более простые задачи, чем возникающие в проблеме выбора, остается справедливым также и для аспектов описания нагрузки в исследованиях по усовершенствованию.

Сравнение исходной системы с ее измененной версией требует, чтобы модель рабочей нагрузки, под управлением которой выполняется

исследование, можно было переносить с исходной системы на измененную. Таким образом, рабочая нагрузка *должна не зависеть от модификации*. Это требование обычно гораздо легче выполнить в работах по усовершенствованию, чем в работах по выбору. Изменения, рассматриваемые в работах по усовершенствованию, обычно малы по сравнению с радикальными различиями между вариантами, сравниваемыми при выборе. Описание низкого уровня даже в терминах физических ресурсов будет, вообще говоря, удовлетворительным при условии, что изменения, которые будут приняты во внимание, не воздействуют на параметры рабочей нагрузки. Например, если мы не исключили возможность замены центрального процессора, наша модель рабочей нагрузки не должна представлять вычислительных запросов заданий на их процессорное время. Следует предпочесть более независимое от ЦП описание, например смесь команд или, еще лучше, смесь операторов языка высокого уровня.

Основная причина этого требования состоит в том, что, как обсуждалось в разд. 6.3.1, сравнение двух версий системы должно производиться при одинаковых рабочих нагрузках, чтобы избежать вариаций вследствие несущественных причин. Этой цели можно достичь, сделав эти две рабочие нагрузки идентичными на уровне потребностей пользователя в обработке информации (см. разд. 6.5). Если бы мы могли предсказать последствия некоторого изменения для коллектива пользователей, этот подход естественно давал бы две различные модели рабочей нагрузки на нижних уровнях. Однако, поскольку достижение этого превосходит наши сегодняшние возможности, используются модели рабочей нагрузки, идентичные на нижнем уровне. Если изменения системы, результат которых нас интересует, не слишком радикальны, две разные модели нижнего уровня отличались бы друг от друга незначительно и предположение об их идентичности в большинстве случаев не окажет сильного влияния на результаты. Можно привести доводы в обоснование того, что обычные приближения, особенно при использовании аналитических методов, достаточно грубы, чтобы игнорировать реакцию коллектива пользователей на изменение в системе. Действительно, удивительное согласие с экспериментальными результатами, полученное для очень примитивных моделей (см. например, примеры 7.9 и 7.13), говорит в пользу такой точки зрения. Эти реакции, однако, нельзя игнорировать, если изменение системы должно повлиять на рабочую нагрузку через эти реакции.

Аргументы, оправдывающие использование моделей рабочей нагрузки низкого уровня (т. е. ориентированных на физические ресурсы) в сочетании с имитационными или аналитическими моделями системы, применимы также к моделям рабочей нагрузки, предназначенным для использования в измерительных экспериментах. В частности, наборы синтетических заданий хорошо приспособлены к большинству измерительных аспектов работ по усовершенствованию. Гибкость синтетических заданий может оказаться полезной, если изменения системы требуют, чтобы менялась модель рабочей нагрузки, или, как упомина-

лось в разд. 7.5, если изменения модели рабочей нагрузки используются для имитации изменений системы. Все, что было сказано выше относительно выбора уровня, на котором будут характеризоваться задания, следует иметь в виду в исследованиях по усовершенствованию также и при проектировании исполняемых моделей рабочей нагрузки, а не только для аналитических или имитационных моделей.

Задачи

7.1. Исследуйте форму функции $\overline{T}(\mu_1, \mu_2)$ для модели центрального обслуживания с двумя обслуживающими приборами при $b_{11}=0.1$, $b_{12}=0.9$ и $N=3$. Начертите на плоскости (μ_1, μ_2) некоторые ее линии и укажите области, где μ_1 порождает узкое место, где μ_2 порождает узкое место, и области, где модель сбалансирована. Если задано некое значение μ_1, то существует ли такое μ_2, которое оптимизирует $\overline{T}$?

7.2. Вам предоставлена система, конфигурация оборудования которой состоит из ЦП, основной памяти, одного дискового контроллера и дискового устройства, одного ленточного контроллера и лентопротяжного устройства и одного контроллера ввода-вывода, управляющего устройствами ввода перфокарт и печати. Вам также предоставлен набор синтетических заданий, который составляет удовлетворительную модель рабочей нагрузки данной установки. Синтетические задания принадлежат типу ориентированных на физические ресурсы, т. е. они сконструированы так, как описано в примере 5.8. Спланируйте серию измерительных экспериментов, основанных на использовании этой модели рабочей нагрузки и нацеленных на определение ресурса или ресурсов, создающих узкое место, и на улучшение средней пропускной способности системы методом последовательного устранения обнаруженных узких мест.

7.3. Следуя концептуальному подходу, описанному в разд. 7.3.2, определите узкие места для модели вычислительной системы, имеющей вид открытой сети. Возьмите в качестве первичного индекса среднее время прохождения $\overline{t}$, формула для которого в экспоненциальном случае приведена в (4.86). Предположим, что система включает ЦП, барабан и диск, обладающие значениями параметров, приведенными в табл. 7.2, и обслуживающий прибор ввода-вывода с $\mu_4=5\ с^{-1}$. Кроме того, предположим, что средняя интенсивность поступления составляет $\lambda=2$ задания/с и что эти задания завершаются только после обслуживания их диском (т. е. $b_{21}=1$). Используя модель с рис. 4.16, определите, какой из обслуживающих приборов вызывает появление узкого места в системе. Сравните ваши результаты с результатами примера 7.1, где подобная система моделировалась замкнутой сетью с очередями.

7.4. Используя кривые рис. 7.12 и значения параметров, для которых они были нарисованы (см. пример 7.3), постройте поточечно соответствующие $\overline{T}(N)$ и $\overline{t}(N)$. Покажите, что они совпадают с кривыми (c) и (d) на рис. 7.10.

7.5. Проведите аналитическое исследование модели открытой сети, построение которой рассматривалось в задаче 7.3, чтобы увидеть, выполняются ли следующие заключения, полученные в разд. 7.3.2:

a) если все коэффициенты использования равны между собой, то система, вообще говоря, не сбалансирована, и наоборот;

b) если изменимы только b_{12} и b_{13} производительность имеет оптимум, соответствующий определенной паре значений этих параметров;

c) точка оптимальной производительности, если она существует, вообще говоря, не совпадает с точкой баланса или с точкой равномерного использования.

7.6. Обобщите результаты примера 7.5, учитывая перекрытия пар и троек ресурсов в модели центрального обслуживания [уравнения (7.18 — (7.24)] на случай модели с s обслуживающими приборами. Попытайтесь выразить вероятности перекрытия $\rho_{ij\ldots k}$ как функций коэффициентов использования $\rho_i, \rho_j, \ldots, \rho_k$, вероятностей перекрытия низшего порядка $\rho_{ij}, \ldots, \rho_{hk}, \rho_{ijh}, \ldots$ и нормировочной функции $G(N)$.

7.7 Попытайтесь построить график Кивиата для системы, исследованной в задаче 7.3. Выберите переменные, которые будете откладывать по осям (число осей не

обязательно ограничено восемью) так, чтобы облегчить диагностику системы, в частности обнаружение узких мест. Сформулируйте гипотезы об усовершенствовании, основанные на построенном вами графике Кивиата. Определите, какой дополнительный анализ необходимо провести для подтверждения справедливости гипотез, и если потребуются дополнительные данные для их проверки, то какие.

7.8. Постройте график Кивиата для системы с равномерным использованием ресурсов, параметры которой имеют значения, приведенные в табл. 7.3, кроме μ_3, равного $16.66\ с^{-1}$. Отложите по осям значения тех же переменных, что и на рис. 7.14. Используя те же соглашения, нарисуйте график Кивиата той же системы в точке оптимума $\overline{T}$, т. е. для $b_{12}=0.85$ и $b_{13}=0.1$. Сравните эти два графика и прокомментируйте полезность графика Кивиата как средства грубого усовершенствования системы и как средства тонкого усовершенствования.

7.9. Используя профиль использования системы, подобный рис. 7.13, сформулируйте гипотезы о способах усовершенствования системы с виртуальной памятью, описанной в примере 7.3 (отметим, что $\mu_3=16.66\ с^{-1}$), при $N=5$. Попытайтесь предсказать с помощью подходящего уравнения времени системы влияние на $\overline{T}$ увеличения на 20% скорости вращения страничного барабана (при этом увеличивается $R_{бр}$ и уменьшается $t_{сбр}$).

7.10. Повторите задачу 7.9, предполагая $N=2$.

7.11. Проанализируйте концептуальные взаимоотношения между двумя определениями узкого места, содержащимися в (7.3) и (7.25), в случае модели рис. 7.5. Предположите, что параметры заданий $\omega_1, \ldots, \omega_v$, использованные в определении (7.25), представляют глобальные запросы времени ЦП, времени барабана и времени диска на каждое задание. Кроме того, сделайте в (7.3) индексом производительности $\overline{t}$ вместо T и замените d и ω_i в (7.25) их средними значениями по всем заданиям.

7.12. Наши вычисления в примере 7.10 показали, что в результате смены стратегии распределения памяти поровну в модели примера 7.3 на стратегию распределения памяти с переменными разделами, основанную на (7.38), при $N=3$ ожидается повышение производительности приблизительно на 5%. Каково будет ожидаемое улучшение при $N=5$?

7.13. Повторите задачу 7.12, предполагая, что $\mu_3=8.33\ с^{-1}$, как в табл. 7.3. Вычислите также ожидаемое улучшение $\overline{T}$ и $\overline{t}$ для этого значения μ_3 при $N=3$, как в примере 7.10.

7.14. Мультипрограммная система содержит один ЦП, один барабан и один диск. Параметры модели центрального обслуживания для этой системы таковы: $\mu_1=100\ с^{-1}$, $\mu_2=50\ с^{-1}$, $\mu_3=20\ с^{-1}$, $b_{12}=0.6$, $b_{13}=0.3$ и $N=5$. Нужно повысить производительность этой системы, представляемую ее $\overline{T}$. На попытку усовершенствования можно истратить самое большее 50 000 долларов. Для простоты предположим, что b_{12} и b_{13} можно менять бесплатно (сохраняя их сумму постоянной), а величинам μ_1, μ_2 и μ_3 можно дать непрерывные приращения со следующими ценами: $\Delta c_1/\Delta\mu_1=1400$ долларам, $\Delta c_2/\Delta\mu_2=3200$ долларам и $\Delta c_3/\Delta\mu_3=1000$ долларам. Кроме того, предположим, что размер памяти — неизменяемый параметр. Используя кривые рис. 7.16 или эквивалентный аналитический подход, какие суммы вы бы затратили на увеличение μ_1, μ_2 и μ_3. Зависит ли окончательное значение $\overline{T}$ от того, определены ли изменения величин μ до или после изменений b_{12} и b_{13}? Имеет ли смысл оптимизировать $\overline{T}$, маневрируя сначала значениями b_{12} и b_{13}, а затем увеличивая μ_i и вновь повторяя оптимизацию?

7.15. Как бы вы оценили емкость системы, о которой вы знаете только ее профиль использования? Предположим, что задан профиль, представленный на рис. 7.13(a). Используя в качестве примера профиль рис. 7.13 (b), соответствующий максимальному достижимому значению $\overline{T}$ при изменении только b_{12} и b_{13}, попытайтесь найти подход к приблизительной оценке емкости. Проверьте ваши соображения на профиле рис. 7.15 (a) и данных табл. 7.5.

7.16. Для проблемы развития, обсужденной в примере 7.11, определите, что будет более удобным с точки зрения потенциального дальнейшего увеличения $\overline{T}$ при настройке: увеличение размера памяти до $128K$ слов или замена барабана на более быстрый. Используйте приближения, введенные в конце разд. 7.4.1 для вычисления ем-

кости. Прокомментируйте величину ожидаемых вами ошибок при использовании этих приближений в вычислениях, необходимых в этой задаче.

7.17. Постройте профиль использования системы, изученной в примере 7.11, и вычислите, используя подходящее уравнение времени системы, увеличение $\overline{T}$, которое произойдет вследствие увеличения μ_1, μ_2, μ_3 согласно табл. 7.8. Сравните полученные результаты с приведенными в табл. 7.8, вычисленными с помощью модели центрального обслуживания, и обсудите причины всех заметных несоответствий, которые вы сможете обнаружить.

7.18. Что бы вы подумали о процедуре самонастройки, которая основана на управлении степенью мультипрограммирования в системе с виртуальной памятью по результатам периодических измерений длин очередей к страничному барабану? Предположим, что страничный барабан имеет восемь секторов и по одной очереди к каждому сектору. Процедура определяет периодически сумму длин очередей и выбирает значение N на следующий интервал времени в соответствии со значением этой суммы. Придумайте разумные методы для

a) выбора N (связали бы вы с каждым диапазоном длин очередей определенное значение N или позволили бы N уменьшаться и увеличиваться не более чем на 1 при каждом принятии решения, или приняли бы какую-нибудь еще методику?);

b) выбора задания (заданий), которые следует вывести из работы, когда надо уменьшить N;

c) распределения освободившейся области основной памяти между заданиями, еще находящимися в работе после уменьшения N;

d) определения подходящей длины выборки.

7.19. В примерах 7.3 и 7.10 мы рассматривали страничный барабан того же типа, что и барабан, использованный в обсуждавшейся пакетной версии ПРОП-системы, например в примере 7.1, т. е. барабан с единственной очередью запросов, которые должны быть обслужены в соответствии со стратегией «первым пришел — первым обслужен». На практике страничный барабан организуется иначе из-за требований к скорости, налагаемых системой виртуальной памяти на второй уровень ее иерархии памяти. Барабан делится на секторы, и страница может быть передана на барабан или с него за время прохождения под магнитными головками одного сектора. Для каждого сектора ведется отдельная очередь, так что один страничный обмен может произойти в каждом секторном интервале времени. Определите, каких изменений кривой толкотни для системы вы ожидаете при замене исходного страничного барабана на только что описанный. Можно ли модель центрального обслуживания, использованную в примерах 7.3 и 7.10, применить для количественного предсказания этих эффектов? Позволили бы нам изменения, которые необходимо сделать в модели, все же получить решение в замкнутой форме?

8
Оценка производительности в задачах проектирования

8.1. ЗАДАЧИ ПРОЕКТИРОВАНИЯ

Третьей категорией задач, перечисленных в разд. 1.5, к которым применимы методы оценки производительности, является категория задач проектирования. Границы этой категории так же нечетки, как и остальных. Определение, данное в разд. 1.5, включает проблемы, возникающие при попытке определить, как должна быть сконструирована вычислительная система, способная удовлетворительно решать задачи переработки информации. Коренной особенностью проектирования в отличие от выбора является его созидательный характер. Выбор ограничен тем, что имеется в данный момент на определенном участке рынка, тогда как проектирование состоит в создании новой системы, отвечающей тем требованиям пользователя, которые существующие системы удовлетворить не могут. Разумеется, проектирование системы не означает, что система должна быть создана из ничего. Как правило, конструктор использует должным образом уже существующие компоненты, которые будут комбинироваться по-новому для удовлетворения новых требований или же старых требований, но более подходящим образом. Эти компоненты должны быть выбраны среди имеющихся. Если не существует компонента, способного выполнять определенную функцию в соответствии с пожеланиями конструктора, придется спроектировать и изготовить новый компонент, опять же используя, если возможно, существующие компоненты.

Проблемы усовершенствования, как и проблемы выбора, относятся к законченным изделиям. Однако процесс изменения системы с целью ее улучшения похож на процесс проектирования системы. Оптимизация является важным аспектом проектирования, а проектирование систем, которые потом придется приспосабливать к особенностям их операционной среды, является обычной практикой в создании вычислительных систем (см. разд. 7.2 и 7.4.1). Не только проектные работы часто сопровождаются работами по выбору и усовершенствованию, а иногда и потребность в некоем проектировании вытекает из результатов усовершенствования или выбора. Например, в некоторой организации может быть принято решение спроектировать конфигурацию своей установки, если работа по сравнительному выбору не дала приемлемых результатов, или спроектировать новую операционную систему для существующей конфигурации, если работа по усовершенствованию показала удобство такого решения.

Все проблемы проектирования можно в принципе сформулировать следующим образом: создать *исполнимое описание* системы, удовлетворяющей заданным *проектным спецификациям*. Результатом работы по проектированию является описание, понятное для группы изготовителей и используемое ими для фактического изготовления системы. Это описание проектировщики готовят на основе проектных спецификаций, которые могут быть сформулированы как самими проектировщиками, так и некоторым «заказчиком» или ими совместно. В случае вычислительных систем, как и для всех технических изделий, существуют три типа спецификаций: *функциональные спецификации*, *спецификации производительности* и *спецификации стоимости*. Вообще, как мы видели в разд. 1.4, функциональные спецификации относятся к тому, что, как полагают, должна уметь делать система (например, правильно обрабатывать информацию, поступающую в систему на определенном носителе, запоминать ее, выдавать результаты в определенной форме, взаимодействовать со своими пользователями заданным образом, обеспечивать секретность и сохранность своей информации и т. д.). Спецификации производительности предписывают, насколько хорошо должны выполняться функции, определенные функциональными спецификациями (например, насколько быстро, насколько эффективно, насколько дешево). Наконец, спецификации стоимости обычно сводятся к требованию, чтобы общая стоимость системы была минимальна при выполнении ограничений, наложенных остальными типами спецификаций. Следует отметить, что общая стоимость включает цены компонентов оборудования и программного обеспечения наряду со стоимостью проектирования и изготовления системы. Если система проектируется для использования внутри фирмы, общая стоимость должна включать стоимость эксплуатации и технического обслуживания.

Очень широкий смысл, который можно придать термину *производительность*, уже комментировался в разд. 1.1. Здесь мы только повторим что под спецификациями производительности мы будем понимать только те из них, которые можно выразить в терминах нашего ограниченного определения производительности, т. е. типами индексов, введенными в разд. 1.4. Таким образом, говоря о спецификациях производительности, мы будем иметь в виду утверждения, подобные следующим: средняя скорость выполнения команды в системе должна быть выше определенного минимума; средняя пропускная способность при обработке указанной рабочей нагрузки должна быть по крайней мере равна заданному значению и не должна падать больше, чем на определенный процент при определенной пиковой рабочей нагрузке; среднее время ответа на команды редактирования и его стандартное отклонение не должны превышать определенных максимальных значений; число активных терминалов, работающих в режиме редактирования, которые система может обслужить при заданном среднем времени ответа, должно быть больше заданного минимума и т. д.

Эти спецификации производительности очень напоминают те, которые включаются в заявки, подготавливаемые для поставщиков при

конкурентном выборе (см. разд. 6.2.1). Фактически заявки являются основой для проектной разработки, выполняемой каждым поставщиком с целью удовлетворения будущих потребностей заказчика. Это пример проектной разработки, предпринимаемой ради специализированной установки. Как мы уже видели в гл. 6, рабочая нагрузка такой установки может быть известна с большей или меньшей степенью подробности, но все-таки всегда относительно хорошо известна, чего нельзя сказать о рабочей нагрузке, для которой проектируются установки в ряде других случаев. Часто рабочая нагрузка, которую предстоит обрабатывать системе, известна в момент проектирования настолько приблизительно, что не может быть указана точнее, чем в категориях широкой области применения (например, научные вычисления, обработка экономических данных) и необходимых «размера» или «мощности» системы (характеризуемых прилагательными «большой», «средний» или «малый»). В этих случаях количественные спецификации производительности, подобные приведенным в предыдущем абзаце, которые должны явно оценивать приемлемость данной рабочей нагрузки, либо невозможно использовать, либо нужно определять для некоторой фиктивной рабочей нагрузки, которая, по мнению проектировщиков, будет достаточно полно представлять возможные применения системы.

Только что описанная ситуация возникает, если приходится проектировать систему не для единственной специфической установки или семейства похожих установок, а для большого числа применений, свойства которых можно охарактеризовать таким всеобщим описателем, как «большие научные и инженерные вычисления». Бóльшая точность требует знания каждой отдельной установки, на которой будет работать система, или по крайней мере достаточно жесткого ограничения класса установок, для которого проектируется система. Это означает, что пришлось бы проектировать и изготавливать гораздо большее количество гораздо более специализированных компьютеров, что, очевидно, неудобно для изготовителя. С другой стороны, проектирование системы, копии которой, как полагают, должны работать удовлетворительно на установках с совсем различными характеристиками, не может основываться на таких грубых и поверхностных определениях ее рабочей нагрузки. Действительно, систему, предоставляющую все, что угодно, любым пользователям, даже представить невозможно.

Способом разрешения такого противоречия является то, что мы могли бы назвать (отвлекаясь от некоторых реальных сложностей) *подходом в три этапа* (рис. 8.1). На первом этапе изготовители проектируют не систему, а *семейства компонентов*. В них входят один или несколько ЦП, различные устройства основной и внешней памяти, оборудование связи, каналы ввода-вывода, контроллеры устройств и т. д. Некоторые из этих компонентов могут быть даже приобретены у других изготовителей вместо самостоятельной разработки, усовершенствования и изготовления внутри фирмы. Цели семейства компонентов обычно не устанавливаются точно. В самом деле, эти компоненты явля-

ются «кирпичиками» для второго этапа, этапа *выбора варианта установки в соответствии с требованиями заказчика*. Второй этап характеризуется гораздо более детальным знанием рабочей нагрузки, обусловленным тем, что теперь система собирается для специфической установки. На этом этапе можно определить индексы производительности, а удовлетворение требований производительности может войти в число целей

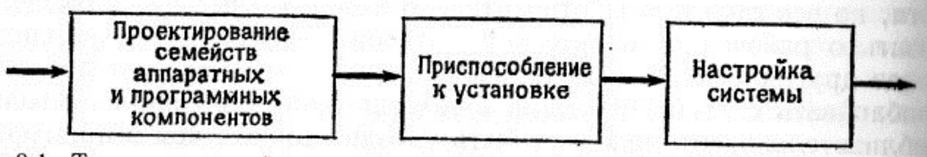


Рис. 8.1. Трехступенчатый подход к проектированию вычислительной системы. Знание о специфической рабочей нагрузке, которую предстоит обрабатывать системе, существенно возрастает по мере продвижения от первого этапа к третьему.

проектирования. Это проектирование выполняется либо заказчиком, либо изготовителем и обычно приводит к выбору аппаратных и программных компонентов для системы. Однако неполнота спецификаций рабочей нагрузки и неадекватность методов проектирования делает необходимым третий этап. Этот этап, называющийся *настройкой системы* (см. разд. 7.4.1), выполняется под управлением реальной рабочей нагрузки после установки системы.

Такой трехэтапный подход, принятый на первых порах развития вычислительной техники и все еще широко использующийся сейчас, влияет как на проектирование программного обеспечения системы, так и на разработку аппаратных компонентов и конфигураций. По причинам, приведенным выше, операционная система разрабатывается не для отдельной, специализированной установки, а для большого класса установок. Некоторые операционные системы обеспечивают различные режимы обработки, а в большинстве случаев они способны управлять рядом различных конфигураций, которые можно построить, комбинируя компоненты данного семейства. Необходимая гибкость часто получается в ущерб эффективности. Излишняя универсальность приводит к увеличению объема операционной системы, большей стоимости ее проектирования, написания и отладки, к большему расходу ресурсов, к меньшей надежности по сравнению с системой, приспособленной к конкретной конфигурации и к конкретному применению. С другой стороны, возможность использования ее для большого количества установок, позволяющая распределить затраты на нее между этими установками, экономически оправдывает такую практику.

В этой главе мы будем по большей части заниматься вторым этапом этого подхода. Первый этап заключает в себе проектирование компонентов оборудования, проблему весьма интересную с точки зрения технологов и инженеров, но не с точки зрения производительности вычислительных систем. Предполагается, что характеристики производительности компонентов удовлетворяют пожеланиям разработчиков системы и совместимы с существующими экономическими и технологическими ограничениями. Разработчики системы часто играют важную роль в проектировании компонентов. Они не только используют имеющиеся компоненты в качестве «кирпичиков», но и стимулируют своими требованиями создание новых и более удобных узлов оборудования. Среди компонентов оборудования центральные процессоры требуют наибольшего внимания со стороны разработчиков системы в основном из-за их роли в выполнении программ, составляющих как рабочую нагрузку, так и операционную систему. Однако, несмотря на его влияние на производительность системы, проектирование ЦП здесь не обсуждается. Мы только рассмотрим применения оценки производительности к проблемам комбинирования различных компонентов и организации их работы так, чтобы удовлетворить некоторым заданным спецификациям производительности тех типов, которые встречаются на втором этапе. Наше рассмотрение будет включать некоторые аспекты производительности в проектировании операционной системы, хотя эту деятельность следует считать завершенной на первом этапе. Мы не будем касаться настройки, поскольку она уже обсуждалась в разд. 7.4.1.

8.2. МЕТОДЫ ПРОЕКТИРОВАНИЯ

В разд. 8.1 мы выделили трехэтапный подход, который часто принимается для проектирования и изготовления вычислительных систем. Вместо того чтобы создавать системы для специализированных установок «от самого начала», изготовители компьютеров комплектуют наборы компонентов, которые затем используются ими же, а иногда и самими заказчиками для конструирования систем с конкретными характеристиками. Противоположный подход, состоящий в проектировании компонентов после того, как система специфицирована в деталях для данной установки, используется только в специальных случаях. Он главным образом ограничивается некоторыми компонентами (такими, как блоки системного программного обеспечения или оборудования, отсутствующие на рынке), отдельными научными исследованиями или одноразовыми применениями.

В любом случае независимо от того, принят трехэтапный подход или нет, существуют две принципиально различные методики проектирования. Для определения этих методик можно использовать различные термины. Первая называется проектированием *снизу вверх*, *горизонтальным*, *изнутри наружу* или *итеративным*, а вторая — проектированием *сверху вниз*, *вертикальным* или *снаружи внутрь*, или *иерархическим*. Ни один из этих терминов не является вполне удовлетворительным, но, поскольку приходится делать выбор, мы выберем прилагательные «итеративное» и «иерархическое». Проектирование основывается на одной из этих двух методик или, гораздо чаще, на смеси обеих в разных пропорциях.

Итеративное проектирование состоит в выработке первоначального решения проблемы и в преобразовании его в приемлемое решение ите-

ративными изменениями. Таким образом, проектирование рассматривается как процесс проб и ошибок, который через последовательные приближения приводит к системе, удовлетворяющей заданным спецификациям. Цикл проектирования представлен на рис. 8.2. Этот рисунок отражает некоторые проблемы, которые мы кратко обсудим, подчеркивая вопросы производительности.

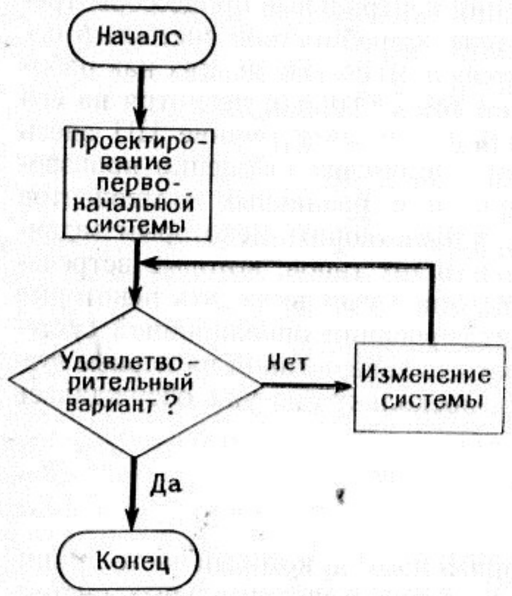


Рис. 8.2. Блок-схема методологии итеративного проектирования.

Во-первых, как проектировать первоначальную систему? В спецификациях проектирования содержатся в сжатом виде необходимые требования. Обычно проектировщик на этом этапе использует знание характеристик существующих компонентов или технологических приемов, а также решений, принятых в сходных установках. Как мы увидим в разд. 8.3.1, аналитические модели, даже крайне простые, могут оказаться весьма полезными. Прежде всего нам желательно разумное приближение к приемлемой конфигурации. Выбор первоначального решения часто оказывает глубокое влияние на результат процесса проектирования и даже может сделать его невозможным или очень дорогим. Таким образом, очень важно, чтобы этот выбор был сделан с большой осторожностью и чтобы были использованы доступные методы и средства для увеличения вероятности успеха. Когда первоначальная конфигурация определена, можно выбрать или разработать для нее операционную систему. Первоначальная система «оборудование — программное обеспечение» должна быть определена достаточно детально, чтобы сделать итеративный анализ производительности возможным и целесообразным.

Во-вторых, как определить, удовлетворительна ли полученная система? Эта проверка должна быть выполнена для всех спецификаций, включая функциональные и экономические. С точки зрения производительности, проект должен быть проверен с помощью некоторого метода анализа производительности. Пока система не изготовлена в некотором виде, методы измерения применить нельзя. Следовательно, придется использовать аналитическое или имитационное моделирование. Модель (или модели) рабочей нагрузки, которая будет управлять моделью системы, так же как и интересующие нас индексы производительности, следует получить на основании спецификаций производительности. Несмотря на высокую стоимость имитационного моделирования, ему

часто отдают предпочтение по сравнению с аналитическими методами из-за присущей ему гибкости и большей степени детальности. Вероятно, только лишь изготовитель машин обладает ресурсами, необходимыми для выполнения анализа производительности измерительными методами с самых ранних этапов разработки. Изготовители машин обычно имеют в наличии все компоненты, которые могут быть относительно легко соединены друг с другом. Кроме того, имеются или могут быть легко приобретены или изготовлены измерители и средства сжатия данных. Модели рабочей нагрузки в виде набора эталонов, подготовленных заказчиками, или даже в виде исполняемых трассировочных записей, полученных самими разработчиками, также обычно имеются в наличии. Последующие разделы и в особенности разд. 8.5 содержат некоторые дополнительные детали, относящиеся к использованию различных методов оценки в проектировании систем.

В-третьих, как определить изменения, которые необходимо сделать в проекте, не удовлетворяющем спецификациям? Читатель, конечно, уже заметил сходство между итеративным проектированием и итеративным усовершенствованием (сравните, например, рис. 8.2 с рис. 7.1). Если мы будем рассматривать только требования производительности, то можем сказать, что процесс итеративного проектирования состоит из усовершенствования первоначальной версии системы с помощью последовательных изменений, которые делаются для выполнения заданных спецификаций. Таким образом, бóльшая часть нашего обсуждения методов усовершенствования в гл. 7, включая диагностику и терапию, применима к этому этапу итеративного проектирования. Основным принципом остается принцип направленного исследования, представление индексов производительности гиперповерхностями в пространстве параметров проекта с целью оптимизации (или по крайней мере улучшения). Формулируются и проверяются гипотезы об усовершенствовании. Результаты проверки дают направление для формулировки будущих гипотез. Работа Нильсена (1967), частично описанная в примере 7.7, как образец применения имитационного моделирования к проблеме усовершенствования, была в сущности работой по итеративному проектированию, так как целью ее было проектирование конфигурации систем с разделением времени.

Иерархическое проектирование основано на противоположном принципе. Структура системы и характеристики ее компонентов выводятся непосредственно из проектных спецификаций, так что результат получается с гарантией функционально корректным, а также удовлетворительным в смысле производительности и стоимости. В чистом виде иерархическое проектирование не требует никаких проверок, кроме, быть может, необходимых для устранения возможных сомнений относительно корректного применения процедур проектирования. С точки зрения производительности этот подход требовал бы в принципе применения методов синтеза производительности (см. разд. 1.4). К сожалению, эти методы, которые позволили бы получать конфигурацию и параметры системы непосредственно из спецификаций производительности, хотя,

возможно, и за несколько шагов, недостаточно развиты, за исключением крайне простых случаев. Однако иерархические методы исследуются и предлагаются с основной целью — обеспечить корректность проектов сложных систем. Иногда можно учесть в этих методах соображения производительности, и в любом случае изучение аспектов производительности этих систем представляет существенный интерес.

Предлагаемые методы заключаются в постепенном уточнении проектируемой системы по иерархии *уровней абстракции*. Как подробно обсуждалось в разд. 8.4, каждый уровень абстракции соответствует описанию системы, характеризующемуся определенной степенью абстрагирования от действительности. Иначе говоря, определенные реальные особенности (например, технологические ограничения, внешние влияния) игнорируются на некотором уровне с целью изучения отдельных вопросов проектирования, на которые отброшенные аспекты, как полагают, не оказывают заметного влияния. Эти особенности затем будут вводиться постепенно на более низких уровнях, так чтобы их влияние могло быть учтено должным образом и проектирование могло продолжаться по этапам дальнейшего уточнения. При переходе от высших уровней абстракции к низшим степень детальности описания системы возрастает, и часто (но не всегда) различные модули, составляющие описание на высшем уровне, оказываются разделенными на связанные друг с другом подмодули на нижнем уровне. В идеальном случае в конце процесса самый нижний уровень должен давать описание, которое можно сразу же использовать для изготовления системы. Это описание должно формулироваться в терминах компонентов оборудования, которые существуют или могут быть спроектированы проектировщиками оборудования, а также в терминах компонентов программного обеспечения, которые могут быть написаны системными программистами.

Заметим, что различие между аппаратурой и программным обеспечением вводится в процедуре иерархического проектирования как можно позже. Это согласуется с основным принципом, не придающим такому различию существенного значения, поскольку границы между аппаратурой и программным обеспечением заметно меняются в ходе технического прогресса. В итеративных проектах различие между аппаратурой и программным обеспечением есть уже в первоначальной системе, хотя некоторые возможности изменения появляются, благодаря наличию этапа изменения в цикле проектирования. В особенности это верно для этапа, предшествующего изготовлению прототипа. Понятно, что как в итеративном, так и в иерархическом проектировании эта гибкость всегда ограничена, если используется только имеющееся оборудование. Однако исследование компромиссов «аппаратура — программное обеспечение» может оказаться весьма плодотворным даже при довольно жестких ограничениях. Еще не настало время, когда процесс проектирования системы как единого целого будет чисто иерархической процедурой без каких бы то ни было ограничений на компоненты. С другой стороны, времена, когда вначале проектировалось оборудование без всякой связи с программным обеспечением, а программное обеспечение «добавлялось» потом, к счастью, прошли навсегда.

Мы уже говорили в разд. 8.1, что существуют три типа спецификаций: функциональные, производительности и стоимости. Какова роль оценки производительности в двух методиках проектирования, которые кратко рассмотрены в данном разделе? Этот вопрос будет изучен с различных точек зрения в оставшейся части этой главы. Однако мы можем здесь повторить общие соображения, высказанные в разд. 6.2.2 относительно роли оценки производительности в проблемах выбора. Производительность в ограниченном смысле является важным, но ни в коем случае не единственным фактором, который следует принимать во внимание при проектировании системы. Воздействие системы на организацию, в которой она будет установлена, обычно нельзя свести к пропускной способности или времени ответа системы. Впечатление, которое она произведет на пользователей (языки, интерфейс, легкость пользования, широта применения, защита конфиденциальной информации), изменения в деятельности организации, которые она внесет, и даже престиж фирмы вследствие самого факта установки системы так же важны, как и значения индексов производительности, а иногда и в большей степени.

8.3. ПРОБЛЕМЫ ПРОИЗВОДИТЕЛЬНОСТИ В ИТЕРАТИВНОМ ПРОЕКТИРОВАНИИ

8.3.1. Аналитические методы проектирования первоначальной конфигурации

Хорошая первоначальная система повышает вероятность успеха итеративной процедуры проектирования. Это также уменьшает количество необходимых циклов, а следовательно, стоимость и продолжительность работы. Однако проектирование первоначальной системы не должно бы быть особенно дорогим или требующим много времени; иначе преимущества, которые мы получаем в итеративной части процедуры, могут обесцениться частично или полностью вследствие возросшей сложности начального этапа. Ясно, что должен быть найден разумный компромисс между двумя конфликтными требованиями о том, чтобы первоначальный проект был «хорошим» и чтобы его можно было выполнить относительно быстро и с небольшими затратами. Как можно достичь обеих целей?

Не существует исчерпывающих и широко используемых процедур для выработки проекта первого приближения вычислительной системы общего назначения. Хотелось бы, чтобы это была последовательность шагов, включающих относительно простые вычисления, из которых следовали бы значения параметров системы и, возможно, соображения о различных организационных аспектах системы (например, схема взаимосвязей оборудования, размещение и организация файлов, схемы управления ресурсами). В других, более устоявшихся областях техни-

ки, существуют приближенные методы такого рода, и проектировщики широко ими пользуются. В вычислительной технике аналогичные методы изобретены и успешно применяются только для тех типов систем, рабочая нагрузка которых легко характеризуется. В таких системах, в особенности предназначенных для использования в реальном времени (например, для резервирования билетов на авиалиниях), рабочая нагрузка, состоящая из одного или немногих типов передачи сообщений, статистически стабильна и относительно предсказуема. Предложенные для подобных систем аналитические процедуры проектирования проводят полное исследование различных проектных компромиссов и возможную оптимизацию индексов стоимость — производительность [см., например, Мартин (1967), Штимлер (1969)]. Из того что систематические методы проектирования существуют только для таких систем, следует сделать вывод, что даже в области проектирования главной проблемой, которую все еще предстоит решать, является проблема описания рабочей нагрузки.

Для всех остальных систем существуют некоторые частные методы, относящиеся только к некоторым аспектам проектируемой системы, и большое разнообразие аналитических моделей, решения которых дают проектировщикам полезные формулы или графики. Большинство этих моделей применимо к компонентам или частям системы или они охватывают только некоторые свойства системы. Пока не построены глобальные модели системы или по крайней мере не собраны согласованные наборы существующих моделей, методы синтеза производительности будут недоступны. Важным шагом в этом направлении является иерархический метод моделирования [см. Браун и др. (1975)]. Однако рассмотренные частные модели, особенно простейшие, можно использовать для проектирования первоначальной конфигурации, как мы покажем на нескольких примерах в этом разделе. К сожалению, как было выяснено выше, полных систематических процедур не существует даже для проектирования первоначальной системы. Таким образом, в наших примерах будут обсуждены некоторые частные, хотя и важные, проблемы. Внимание, как обычно, будет уделено использованию методов оценки производительности, в данном случае аналитических методов.

Пример 8.1. Нужно спроектировать мультипрограммную ПРОП-систему, которая должна использоваться в пакетном режиме обработки с малым временем прохождения на средних и малых установках. Предположим, что проектировщики уже решили, что первоначальная конфигурация будет включать один ЦП, один барабан или диск с неподвижными головками (для нерезидентной части операционной системы, компиляторов программ обслуживания и временных файлов пользователей), один диск с подвижными головками (для буферизации входной и выходной информации и хранения постоянных файлов пользователей), одно устройство ввода с перфокарт и одно печатающее устройство. Барабан и диск должны быть связаны с памятью через два независимых контроллера. В этом примере мы займемся определением приблизительных значений параметров компонентов. Эти значения должны послужить ориентирами при выборе оборудования и проектировании или выборе программного обеспечения системы. Проблема, которую мы здесь рассматриваем, иногда называется *проблемой задания параметров*.

В первом приближении задание параметров ЦП, основной памяти и внешних накопителей для мультипрограммной системы может быть проведено с использованием простой модели системы (например, сеть с центральным обслуживающим прибором с очередями). Так как это замкнутая сеть, то можно моделировать только задания, загружаемые в основную память в некоторый заданный момент времени, а также ресурсы, которые они используют или ожидают. В нашем случае модель будет

Таблица 8.1

Параметры рабочей нагрузки и спецификации производительности для проекта из примера 8.1

		Значение
Параметры рабочей нагрузки [1]		
	$\overline{L}$	0.9×10^6 команд
	$\overline{n}_{\text{ЦП}}$	10
	$\overline{r}_{\text{бр}}$	8000 слов
	$\overline{n}_{\text{бр}}$	6
	$\overline{r}_{\text{дс}}$	12 000 слов
	$\overline{n}_{\text{дс}}$	3
Спецификации производительности		
	$\overline{T}_{\min}$	1 задание/с
	$\overline{t}_{\max}$	6 с

[1] Обозначения см. в табл. 7.2 и 7.3.

состоять из трех обслуживающих приборов (см. рис. 7.5). Степень мультипрограммирования N, предполагаемая постоянной, непосредственно связана с размером основной памяти, как показано в табл. 7.3. Обслуживающие приборы характеризуются своими интенсивностями обслуживания μ_i, которые, как видно из табл. 7.3, зависят от параметров производительности устройств (а также от некоторых параметров рабочей нагрузки).

Анализ потребностей пользователей и ожидаемой рабочей нагрузки мог бы дать нам информацию, показанную в табл. 8.1. Эта таблица содержит значения, которые предполагаются полученными из такого анализа. Разумеется, будут сделаны обычные предположения относительно рабочей нагрузки (статистически одинаковые задания, экспоненциальные распределения). Спецификации производительности для представленной в модели части системы также включены в табл. 8.1. Эти два индекса производительности $\overline{T}$ и $\overline{t}$ не совпадают с соответствующими индексами системы, а характеризуют эту часть системы. Например, $\overline{t}$ есть среднее время нахождения задания (точнее, шага задания) в основной памяти. Заметим, что производительность модели задана неравенствами

$$\overline{T} \geqslant \overline{T}_{\min},\ \overline{t} \leqslant \overline{t}_{\max} \tag{8.1}$$

и что значения $\overline{T}_{\min}$ и $\overline{t}_{\max}$ приведены в табл. 8.1. Нашей ближайшей целью является определение приемлемых значений μ_1, μ_2, μ_3 и N (тем самым определится и размер M основной памяти) так, чтобы были удовлетворены спецификации производительности, а стоимость была минимальна. Для решения этой проблемы перед нами открываются по крайней мере два пути. Один — рассмотреть ряд возможных альтернатив для каждого компонента и вычислить производительность и стоимость для каждого сочетания этих альтернатив. Оптимальное решение будет самым дешевым среди удовлет-

воряющих требованиям производительности. Другой подход состоит в рассмотрении модели стоимости и определении значений параметров, которые минимизируют общую стоимость независимо от того, доступны ли узлы оборудования, характеризующиеся этими значениями. Тогда, как только оптимальное решение будет найдено, мы попытаемся приблизиться к нему, рассматривая только реально имеющиеся компоненты.

Так как применение первого подхода концептуально очевидно, мы кратко проиллюстрируем, как можно работать со вторым. В качестве простой модели стоимости возьмем модель, выраженную законом Гроша (6.11) для обслуживающих приборов, т. е.

$$\mu_i = K_i C_i^2 \quad (i=1, 2, 3), \tag{8.2}$$

и пропорциональное соотношение для памяти

$$C_M = K_M M = K_N N. \tag{8.3}$$

Наиболее серьезным доводом в пользу этих моделей является их простота. Было обнаружено относительное удешевление для скоростей обработки в центральных процессорах (которым пропорционально значение μ_1) и для средних времен доступа и пересылки у барабанов [см. Шарп (1969)]. В частности, в исследовании Шарпа, выполненном методом регрессионного анализа, стоимость устройств с неподвижными головками и вращающимся носителем оказалась пропорциональной $\overline{t}_a^{-0.655} l_{\text{сбр}}^{-0.088}$, где $\overline{t}_a$ — среднее время доступа. С другой стороны, относительное удешевление для размера памяти существенно меньше и им можно пренебречь в (8.3). Заметим, что раз μ_i зависит от рабочей нагрузки, то это же справедливо и для K_i.

Общая стоимость оборудования для моделируемой части системы равна

$$C = h_1 \sqrt{\mu_1} + h_2 \sqrt{\mu_2} + h_3 \sqrt{\mu_3} + h_N N. \tag{8.4}$$

Стоимость C нужно минимизировать. Коэффициенты в (8.4) следует вычислить, исходя из цен и характеристик подходящих коммерческих компонентов, а также из параметров рабочей нагрузки в табл. 8.1. Мы выберем $h_1 = h_2 = h_3 = 10\,000$ долл $\cdot$ с$^{1/2}$ и $h_N = 5000$ долл. (на основе разделения по $16K$ слов, 32 бита на слово).

Из данных табл. 8.1, уравнений в табл. 7.3 и из (7.2) получим $b_{11} = 0.1$, $b_{12} = 0.6$, $b_{13} = 0{,}3$ и

$$N_{\max} = \overline{t}_{\max} \overline{T}. \tag{8.5}$$

Для $\overline{T} = \overline{T}_{\min}$ (мы намерены увеличивать $\overline{T}$ выше этого минимального значения последовательными изменениями) формула (8.5) дает $N_{\max} = 6$.

Чтобы еще упростить задачу, мы уничтожим одно из неизвестных, полагая $\rho_2 = \rho_3$. Хотя мы и узнали в разд. 7.3.2, что оптимум $\overline{T}$ соответствует ситуации, когда более быстрые нецентральные обслуживающие приборы используются интенсивнее, чем более медленные, мы предпочтем отложить поиск оптимума $\overline{T}$ до итеративной части проектирования и этапа настройки. Кроме того, приближения, сделанные нами при оценке значений в табл. 8.1, настолько грубы, что более точный метод оптимизации был бы, вероятно, бесполезен. Мы могли бы также учесть то, что нам известно о ситуациях вблизи максимума $\overline{T}$, и на основании этого выбрать для отношения ρ_2/ρ_3 значение больше 1.

Из $\rho_2 = \rho_3$, (4.97) и (4.108) получим

$$\mu_3 = \frac{b_{13}}{b_{12}} \mu_2, \quad y_3 = y_2 = b_{12} \frac{\mu_1}{\mu_2}. \tag{8.6}$$

Для любого заданного значения μ_1/μ_2 и N мы можем вычислить ρ_1 методом, описанным в разд. 4.3.5. Из каждого значения ρ_1 и заданного $\overline{T}_{\min}$ получим

$$\mu_1 = \frac{\overline{T}_{\min}}{b_{11}\rho_1}; \tag{8.7}$$

μ_2 вычисляется по значению μ_1/μ_2, использованному для вычисления ρ_1, и по μ_3 из (8.6). Стоимость C теперь может быть подсчитана по (8.4). Заметим, что мы свели исходную задачу к задаче только с двумя независимыми переменными, N и μ_1. Если мы для каждого значения N от 1 до $N_{\max}$ [см. (8.5)] построим график $C(\mu_1)$, то получим кривые, изгибающиеся вверх, как показано на рис. 8.3. Существуют два минимума стоимости в которых практически равны. Первый — для $N=3$ и $\mu_2 = 15.57$ с$^{-1}$,

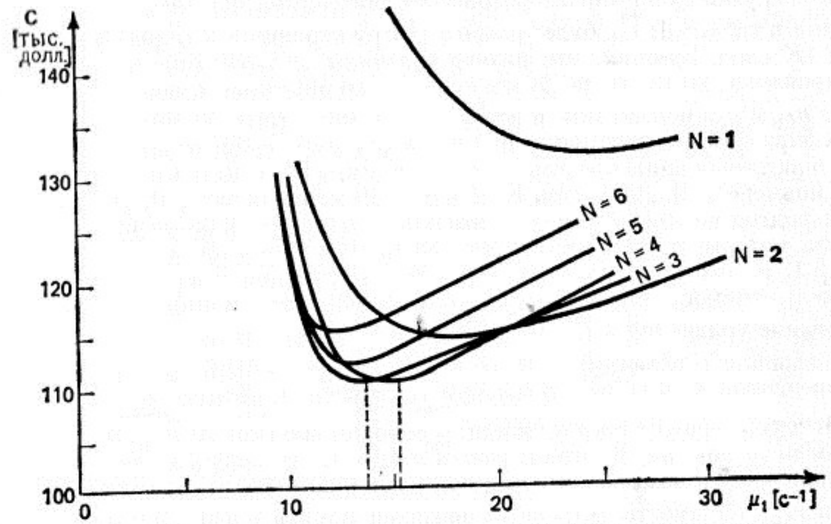


Рис. 8.3. Общая стоимость системы, спроектированной в примере 8.1, как функция от μ_1 и N.

а второй — для $N=4$ и $\mu_1 = 14$ с$^{-1}$. Оба соответствуют $\mu_1/\mu_2 = 1.5$. Видимо, не имеет смысла более точно определять, который из них выбрать, а также анализировать промежуточные случаи вроде $N=3.5$. Поэтому мы произвольно выберем второй, для которого $\mu_2 = 9.33$ с$^{-1}$, $\mu_3 = 4.66$ с$^{-1}$ и $M = 64K$ слов. Мы также получим $\rho_1 = 0.713$, $\rho_2 = \rho_3 = 0.641$. Общая стоимость, согласно (8.5), равна $C = 109\,500$ долларам, из которых на долю ЦП приходится 37 400, на контроллер барабана и сам барабан — 30 500, на контроллер диска и дисковое устройство — 21 500, на основную память — 20 000. Из значений параметров рабочей нагрузки в табл. 8.1 и уравнений в табл. 7.3 мы видим, что следующие характеристики компонентов согласуются с нашими результатами: $\overline{v}_{\text{цп}} = 1.26 \times 10^6$ команд/с, $R_{\text{бр}} = R_{\text{дс}} = 1500$ об/мин, $t_{\text{сбр}} = 10$ мкс/слово, $t_{\text{дс}} = 13$ мкс/слово и $t_{\text{поиск}} = 38$ мс.

Читатель, конечно, уже заметил, что кривые на рис. 8.3 сравнительно пологи. В рассматриваемом диапазоне значений μ_1 общая стоимость изменяется менее чем на 20% от минимальной величины. Это отношение даже меньше в диапазоне значений N для каждого заданного значения μ_1/μ_2. Таким образом, выбор решения не очень критичен, и это подтверждает наше ощущение, что более сложное исследование, чем мы уже предприняли, вероятно, не оправдано.

Исследованная в этом примере задача могла бы быть сформулирована иначе. Вместо назначения границ производительности, аналогичных $\overline{T}_{\min}$ и $\overline{t}_{\max}$, можно было бы задать максимальную стоимость $C_{\max}$ и отыскать среди систем, удовлетворяющих ограничению на стоимость, систему с наибольшим значением $\overline{T}$ или с наименьшим значением $\overline{t}$. Читатель может найти упражнение такого типа в задаче 8.2. ■

Посмотрим теперь, как та же самая модель, которая была использована в примере 8.1, может быть применена к начальному этапу проектирования системы с виртуальной памятью.

Пример 8.2. Конфигурация оборудования ПРОП-системы с виртуальной памятью, которую предстоит спроектировать, состоит из тех же компонентов, что и система, рассмотренная в примере 8.1. Таким образом, можно воспользоваться моделью центрального обслуживания с тремя обслуживающими приборами, как в примере 7.3, в котором исследовалось поведение системы с виртуальной памятью. Предположим, что спецификации производительности этой системы те же, что заданы в табл. 8.1. Рабочая нагрузка будет характеризоваться значениями параметров $\bar{L}$, $\bar{r}_{дс}$, $\bar{n}_{дс}$, приведенными в табл. 8.1; $\bar{r}_{бр}$ будет равно размеру страницы, который мы предполагаем равным $1K$ слов. Заметим, что размер страницы должен определяться в процессе проектирования, но временно будем считать его заданным в постановке задачи. Параметры $\bar{n}_{бр}$ и $\bar{n}_{дс}$ неизвестны, и их предстоит определить. Приняв в качестве модели средних значений для поведения программ функцию времени жизни и предполагая, что это поведение одинаково для всех программ, мы должны также задать a и k [см. (7.11) в примере 7.3]. Допустим, $k=2$ и $a=140$ страницам$^{-2}$. Другой и более реалистичный подход состоит в том, что мы можем измерить кривые времени жизни для программ, которые сочтем типичными для применения этой системы, как описано в разд. 9.3.1, и сконструировать из них «типичную» кривую времени жизни.

Модель стоимости должна отличаться от той, которую описывает формула (8.4). Исследование уравнений в табл. 7.3 убедит читателя, что, так как $\bar{n}_{цп}$ более не является заданной константой, стоимость ЦП не может быть пропорциональна $\sqrt{\mu_1}$. Те же предположения об относительном удешевлении, которые были сделаны в примере 8.1, здесь приводят к стоимости, пропорциональной $\sqrt{\bar{v}_{цп}}$. Кроме того, придется изменить значение h_2, чтобы учесть тот факт, что теперь информация передается между барабаном и памятью страницами, размер которых отличается от предыдущего значения $\bar{r}_{бр}$. Стоимость устройства внешней памяти очень сильно зависит от его размера. В этом и предыдущем примерах мы предполагаем, что размеры барабана и диска известны и постоянны. Проблема выбора этих размеров обсуждается в разд. 8.3.2. Так как M и N теперь не считаются пропорциональными друг другу, стоимость памяти должна быть сделана пропорциональной M. Таким образом, новая модель стоимости будет

$$C=h_1'\sqrt{\bar{v}_{цп}}+h_2'\sqrt{\mu_2}+h_3'\sqrt{\mu_3}+h_M M. \tag{8.8}$$

Мы выберем следующие значения коэффициентов в (8.8): $h_1'=35$ долл$\cdot$с$^{1/2}$, $h_2'=3000$ долл$\cdot$с$^{1/2}$, $h_3'=10\,000$ долл$\cdot$с$^{1/2}$ и $h_M=312.5$ долл/страницу (около 0.95 цента/бит для 32-битовых слов.) Удобный способ организации вычислений состоит в выборе некоторого количества значений y_2 и y_3 [как определено в (4.97)]. Те же аргументы, которыми мы оправдывали произвольное решение при выборе $\rho_2=\rho_3$ в примере 8.1, можно использовать для объяснения, почему мы делаем $\rho_1=\rho_3$ и, следовательно, $y_3=1$. Для каждого значения y_2 мы допускаем, чтобы N изменялось от 1 до N_{max}, и, применяя метод, рассмотренный в разд. 4.3.5, вычисляем соответствующие значения ρ_1. Беря для каждого y_2 максимальное значение ρ_1, мы получим [с помощью (4.108), (7.1), (7.31) и табл. 7.3]

$$\bar{v}_{цп}=\frac{\bar{L}\bar{T}_{min}}{\rho_1}, \tag{8.9}$$

$$\mu_3=\frac{b_{13}\mu_1}{y_3}=\frac{\bar{n}_{дс}\bar{v}_{цп}}{y_3\bar{L}}. \tag{8.10}$$

Затем рассмотрим ряд значений M. Для каждой пары (y_2, M) и значения N, максимизирующего ρ_1 при заданном значении y_2, мы вычисляем $\bar{e}$, используя (7.11). Из (4.97), (7.12) и табл. 7.3 получим

$$\mu_2=\frac{1}{\bar{e}y_2}. \tag{8.11}$$

Здесь, используя (8.8), можно вычислить общую стоимость C. Мы приводим результаты наших вычислений на рис. 8.4. Кривые на этом рисунке показывают C как функцию от N (степени мультипрограммирования, максимизирующей ρ_1 для каждого рассматриваемого значения y_2) и M. Минимальная стоимость соответствует $N=3$ и $M=64K$ слов. Ее значение составляет 98 600 долларов. Параметры, определяющие конфигурацию, равны $\mu_2=27.4$ с$^{-1}$, $\bar{v}_{цп}=1.4\times10^6$ команд/с и $\mu_3=4.66$ с$^{-1}$. Остальные интересующие нас параметры равны: $\bar{n}_{бр}=14$, $\bar{n}_{цп}=18$, $\mu_1=28$ с$^{-1}$, $\rho_1=\rho_3=0.639$ и $\rho_2=0.511$. Из общей стоимости на долю ЦП приходится 41 400, страничного

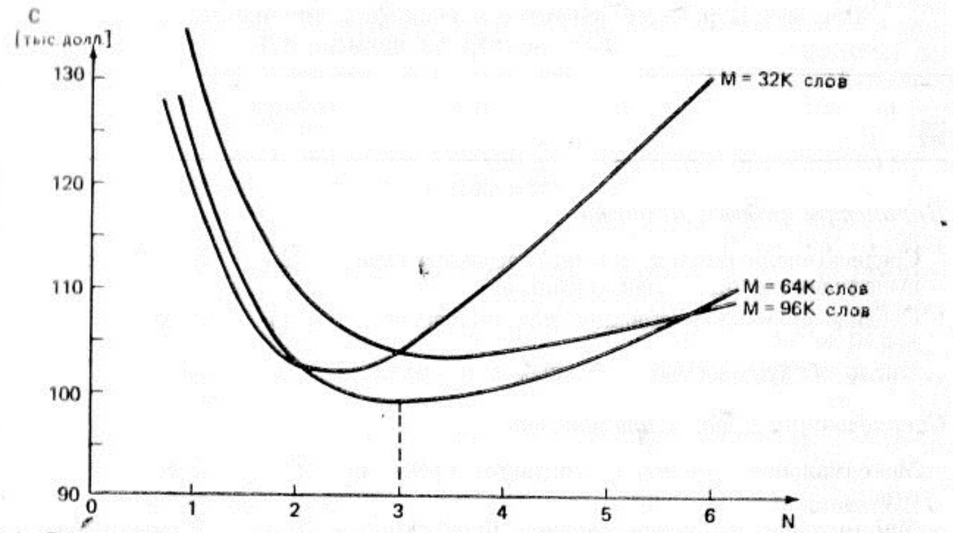


Рис. 8.4. Общая стоимость системы с виртуальной памятью, спроектированной в примере 8.2, как функция от N (степени мультипрограммирования, максимизирующей $\bar{T}$) и M.

барабана — 15 700, диска — 21 500 и основной памяти — 20 000 долларов. Характеристики диска, приведенные в конце примера 8.1, приемлемы и здесь. ЦП должен обладать несколько большей средней скоростью обработки. Для барабана возможными характеристиками могли бы быть $R_{бр}=1500$ об/мин и $t_{сбр}=16$ мкс/слово.

Никаких выводов нельзя сделать из исследований, содержащих грубые, необоснованные приближения (например, в этом примере и примере 8.1). Однако ценным здесь является то, что наша модель системы с виртуальной памятью имеет приблизительно те же стоимость и производительность, что и модель системы, построенная в примере 8.1. В версии системы с виртуальной памятью программистам нет необходимости приспосабливать свои программы к фиксированному размеру раздела ($16K$ слов в примере 8.1) или организовывать их в виде сегментов для попеременной загрузки, по размеру не превосходящих одного такого раздела. Они имеют в своем распоряжении гораздо большее адресное пространство. Однако утверждение, что эти две модели обладают приблизительно равной производительностью, основано на предположении, что автоматическое управление памятью в системе с виртуальной памятью эффективно. В нашем проекте для каждой ситуации выбрано значение N, максимизирующее $\bar{T}$. Если это условие не выполнено, то за увеличение адресного пространства придется заплатить меньшей производительностью или более дорогой системой или и тем и другим.

■

Модели, использованные в примерах 8.1 и 8.2, не подходят для первоначальной конфигурации диалоговой системы. Теперь мы проиллю-

стрируем использование крайне простой модели в проектировании такой системы.

Пример 8.3. Параметры рабочей нагрузки и спецификации производительности диалоговой системы, которую предстоит спроектировать, приведены в табл. 8.2. Если мы предположим, что случайные переменные L и t_i независимы и распределены экспоненциально, мы можем воспользоваться замкнутой моделью с очередями с рис. 4.20.

Таблица 8.2

Параметры рабочей нагрузки и спецификации производительности для проекта из примера 8.3

	Обозначение	Значение
Параметры рабочей нагрузки		
Среднее число команд, которые должны быть выполнены за одно взаимодействие	$\bar{L}$	500 000 команд
Среднее время обдумывания для пользователей	$\bar{t}_t$	20 с
Число обслуживаемых активных терминалов	N	25
Спецификации производительности		
Максимальное значение среднего времени ответа	$\bar{t}_{\max}$	1 с
Минимальное значение средней пропускной способности	$\bar{T}_{\min}$	1 взаимодействие/с

При имеющейся степени детальности нашего описания рабочей нагрузки единственным неизвестным оказывается $\bar{v}_{\text{цп}}$, средняя скорость обработки в ЦП. Заметим, что для использования результатов, полученных в примере 4.10 для модели на рис. 4.20, мы должны предположить, что ЦП, единственный явно представленный компонент системы, предназначен для выполнения процессов в соответствии с дисциплиной разделения процессорного времени.

Из (4.120) и (4.118) получаем

$$\frac{N}{\bar{T}} - \bar{t}_t \leqslant \bar{t}_{\max}, \tag{8.12}$$

$$\frac{N}{\bar{t} + \bar{t}_t} \geqslant \bar{T}_{\min}. \tag{8.13}$$

Так как мы не хотим насыщения системы, мы должны установить $N < N^*$ [см. (4.122)]. Для быстрого вычисления μ_s можно воспользоваться аппроксимацией (4.123), которая дает

$$\mu_s \approx \frac{N-1}{\bar{t}_t} + \frac{1}{\bar{t}}. \tag{8.14}$$

Таким образом, полагая $\bar{t} = \bar{t}_{\max}$ в (8.14), мы получим $\mu_s = 2.2\,\text{с}^{-1}$. Так как $\mu_s = \bar{v}_{\text{цп}}/\bar{L}$, то имеем $\bar{v}_{\text{цп}} = 1.1 \times 10^6$ команд/с.

Если принять аппроксимацию, сделанную в (4.123), можно предположить, что $\bar{t}$ в действительности равно 1 с, и можно вычислить $\bar{T}$ из (4.120) и (4.118). Мы получим

$\bar{T} = 1.19$ взаимодействий/с и $\rho_s = 0.54$. Пригодность аппроксимации можно проверить, вычисляя ρ_s по (4.117). Насыщение, как определил Клейнрок (1968), было бы достигнуто при числе терминалов $N^* = 45$.

Рассматривая более сложные модели, например, заменив представление системы в виде единственного обслуживающего прибора сетью с очередями, можно было бы определить характеристики остальных компонентов. Иерархический подход к моделированию (см. разд. 8.4) можно было бы использовать для постепенного увеличения степени детальности нашей модели. Значение $\bar{v}_{\text{цп}}$, вычисленное здесь, могло бы служить основой для вычисления остальных параметров системы. Или же остальная часть конфигурации могла бы быть выбрана некоторым другим методом (например, по аналогии с известными конфигурациями других сходных систем) и изменялась бы в ходе процедуры итеративного проектирования. В любом случае должно быть введено гораздо более детальное описание рабочей нагрузки, чем в табл. 8.2. ■

8.3.2. Проектирование иерархии памяти

Решающей проблемой в проектировании вычислительных систем является проблема выбора конфигурации *иерархии памяти*. В примерах 8.1 и 8.2 мы рассмотрели только частный аспект этой проблемы. Мы еще вернемся к этим проектам в примерах 8.4 и 8.5, чтобы расширить наше обсуждение, после того как подведем краткий итог причин, по которым иерархии памяти необходимы.

Наличие практически в любой вычислительной системе нескольких типов накопителей, основанных на различных технологических принципах, обусловлено компромиссом между требованиями размера, производительности и стоимости. С момента появления компьютеров, по всей вероятности, еще долгое время этот компромисс будет оставаться единственным разумным решением проблемы, возникающей из конфликта этих требований. В нескольких словах проблема может быть сформулирована следующим образом. В системе для запоминания данных и программ системы и пользователей необходим определенный общий размер памяти $M_{\text{общ}}$. Производительность системы тесно связана с $v_{\text{цп}}$, скоростью обработки информации в центральном процессоре (процессорах). Однако ЦП может работать с полной скоростью только в том случае, когда система памяти может обеспечить необходимую скорость обращений к ЦП. Точнее, время доступа и скорость пересылки данных между системой памяти и ЦП должны иметь тот же порядок величины, что $1/v_{\text{цп}}$ и $v_{\text{цп}}$ соответственно. Так как по соображениям производительности для ЦП требуется память произвольного доступа, то время доступа имеет тот же порядок величины, что и величина, обратная скорости пересылки данных. Следовательно, мы ограничимся рассмотрением времен доступа. Методы изготовления памяти не могут обеспечить время доступа $t_a \approx 1/v_{\text{цп}}$ при такой стоимости, приходящейся на один бит, $c(t_a)$, чтобы общая стоимость системы памяти,

$$C_{\text{общ}} = c(t_a)\, M_{\text{общ}}, \tag{8.15}$$

оказалась допустимой. Пусть на рис. 8.5(а) $M_{\text{общ}} = 10^9$ бит, а $v_{\text{цп}} = 10^6$ команд/с. Таким образом, $t_a \approx 1$ мкс. Из диаграммы рис. 8.6 получим c (1 мкс) ≈ 1 цент/бит. Из (8.15) получим $C_{\text{общ}} = 10^7$ долл. Заметим, что

диаграмма на рис. 8.6 аппроксимирует тренд того, что на самом деле следовало бы изобразить как облако точек, представляющих конкретные системы. С течением времени по мере совершенствования элементной базы это облако смещается в направлении начала координат. Впро-

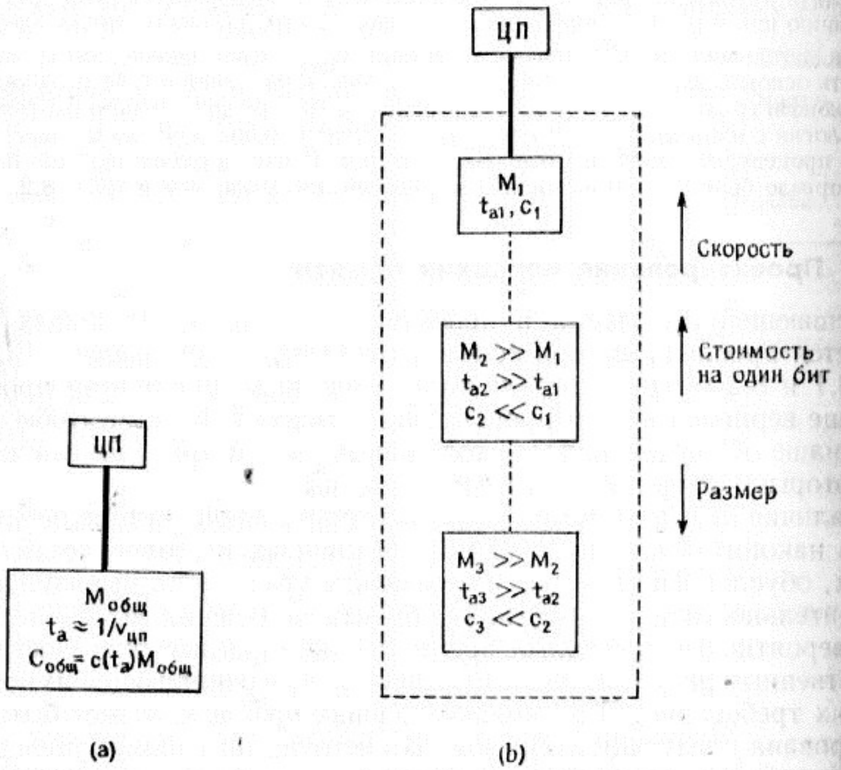


Рис. 8.5. Два варианта реализации одной и той же системы памяти: (a) — одноуровневое решение; (b) — трехуровневая иерархия памяти.

чем, именно поэтому запросы пользователей компьютеров растут со временем.

Отмеченный выше компромисс состоит в изготовлении системы памяти в виде иерархии различных видов памяти. На рис. 8.5(b) показана система памяти, состоящая из трех уровней. Каждый уровень i ($i=1, 2, \ldots$) характеризуется своим размером M_i, своим временем доступа t_{ai} и, следовательно, своей стоимостью на бит: $c_i=c(t_{ai})$. Уровень 1 — основная память, единственная часть системы памяти, непосредственно доступная ЦП. Таким образом, только значение t_{a1} должно иметь порядок величины $1/\nu_{ЦП}$. Более высокие уровни имеют бóльшие времена доступа и, следовательно, существенно более низкие стоимости, приходящиеся на бит (см. рис. 8.6). Таким образом, их делают все обширнее и обширнее по мере снижения цены. Например, если упоминавшаяся выше система памяти изготовлена, как показано на рис. 8.5(b), с $M_1=10^7$ бит, $M_2=10^8$ бит, $M_3=0.9\times10^9$ бит, $t_{a1}=1$ мкс,

$t_{a3}=10$ мс и $t_{a3}=100$ мс, то мы получим

$$M_{общ}=M_1+M_2+M_3\approx 10^9 \text{ бит} \tag{8.16}$$

и

$$C_{общ}=c_1M_1+c_2M_2+c_3M_3\approx 2.9\times10^5 \text{ долл.}, \tag{8.17}$$

так как $c_1\approx10^{-2}$долл./бит, $c_2\approx10^{-3}$ долл./бит и $c_3\approx10^{-4}$ долл./бит. Таким образом, общая стоимость, согласно (8.17), почти на два порядка величины ниже, чем ранее вычисленная для одноуровневой системы

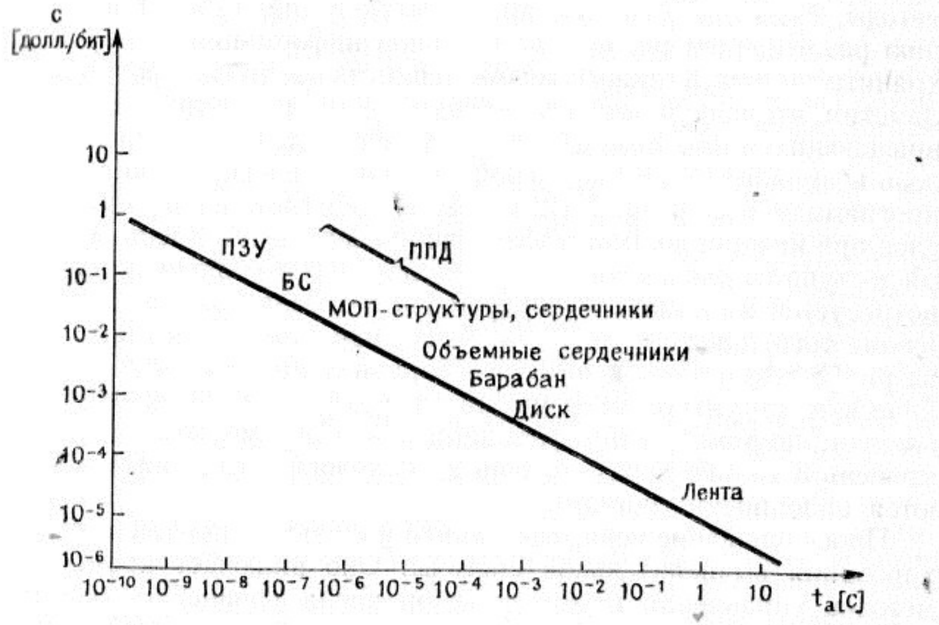


Рис. 8.6. Приблизительное соотношение между стоимостью в расчете на один бит c и временем доступа t_a для памяти на различных элементных базах. [ППД — память произвольного доступа, ПЗУ — постоянное запоминающее устройство, БС — биполярные структуры.]

памяти. Заметим, что в нашем обсуждении мы пренебрегли важным физическим ограничением. Время доступа высокоскоростной памяти зависит от ее размера. Таким образом, даже если забыть об экономических факторах, остаются еще технические причины для ограничения размера основной памяти, и максимальный размер будет убывать с ростом скорости ЦП. Однако экономические факторы обычно ограничивают размер памяти прежде физических.

Итак, мы приходим к выводу, что иерархическая структура существенно уменьшает общую стоимость системы памяти и делает ее приемлемой. Но всегда ли она эквивалентна одноуровневому решению в терминах производительности? Чтобы ответ на этот вопрос был утвердительным, должно быть выполнено следующее условие: ЦП всегда находит в основной памяти нужную информацию в тот момент, когда в ней возникает необходимость. Другими словами, при иерархической

системе памяти для центрального процессора память окажется как бы состоящей из одного уровня размера $M_{общ}$ и в идеальном случае с временем доступа t_{a1}. Если ни одно обращение к уровню 1 не окажется успешным и вся информация должна быть получена, например, с уровня 2, время доступа к системе памяти с точки зрения ЦП будет приблизительно равно t_{a2}, а производительность системы катастрофически упадет. Мы можем сказать, что иерархия памяти предоставляет ЦП *эквивалентное время доступа* t_{ae}. Сделать t_{ae} по возможности близким к t_{a1} — главная задача методов управления памятью. Эти методы, будь они ручные или автоматические, относятся к управлению размещением различных массивов информации, которые должна хранить система, а также к перемещению их на другие уровни иерархии. Заметим, что использование на всех уровнях иерархии, кроме первого, вращающихся накопителей или устройств с передачей массивов оказалось возможным благодаря свойству локальности, обычно присущему программам (см. разд. 7.4.2 и 9.2.2). Действительно, передача блока слов при каждом доступе эффективно распределяет очень большое время доступа на много слов, но только потому, что большинство этих слов потребуется в ближайшем будущем вследствие свойства локальности. Кроме того, следует заметить, что связи между уровнями иерархии на рис. 8.5(b), обозначенные пунктирной линией, представляют иерархическую структуру системы памяти только графически. Из них не следует, например, что информация может перемещаться с уровня 1 на уровень 3 только через уровень 2. Топологические связи представляются сплошными линиями.

Проектирование иерархии памяти является основной частью проектирования вычислительной системы. Оно включает кроме указания методов управления памятью выбор числа уровней и схемы связей между уровнями наряду с технологическим типом устройства, размером и характеристиками производительности для каждого уровня. Мы теперь еще раз кратко обсудим проекты первоначальных конфигураций, описанные в примерах 8.1 и 8.2, и обсудим их с точки зрения проектирования иерархии памяти.

Пример 8.4. В ПРОП-системе пакетной обработки из примера 8.1 было с самого начала решено, что система памяти будет состоять из трех уровней: основная память произвольного доступа, барабан и диск. Оба вращающихся устройства будут соединены с основной памятью и будут пересылать блоки информации в нее и из нее по директивам операционной системы. Таким образом, используемые типы устройств на этих трех уровнях были более или менее выбраны с самого начала. Метод управления памятью был также выбран без рассмотрения какой-либо другой альтернативы. Каждая программа, будучи загружена в основную память, занимает раздел фиксированного размера. Управление этим пространством памяти выполняется программистами посредством соответствующих карт управления заданиями или операторов языка высокого уровня, которые компилятор транслирует в вызовы программ распределения памяти или в обращения к операционной системе. Программы во время исполнения нуждаются в доступе к барабану и диску, так как не вся информация, которую им предстоит обработать или которую они порождают, обычно может разместиться в разделе памяти. Эта информация может включать часть самих программ, входные данные, выходные результаты, файлы и библиотечные подпрограммы, к которым им необходим доступ, и нерезидентные модули операционной системы. Так как время доступа к барабану меньше, чем к диску, мы будем стремиться размещать на барабане информацию, используемую чаще. С другой стороны, из-за более высокой стоимости, приходящейся на бит, барабан будет сделан меньшего размера, чем диск.

Наша процедура проектирования, основанная на модели с центральным обслуживающим прибором, явно определила размер основной памяти M (который ради согласования с обозначениями рис. 8.5 (b) следовало бы назвать M_1) и скорости барабана и диска. Точнее, мы вычислили μ_2 и μ_3, являющиеся функциями времени доступа и скорости передачи этих двух вращающихся устройств. Как определить размеры M_2 и M_3 и скорость основной памяти? Какую роль играют эти параметры в проекте?

Порядок величины t_{a1}, времени доступа к основной памяти, задается выражением $1/\bar{v}_{цп}$, а порядок величины скорости передачи — значением $\bar{v}_{цп}$. Лучшие оценки этих параметров памяти можно получить, если известна общая интенсивность обращений к памяти. Это требует знаний архитектуры ЦП, смеси команд и скоростей передачи барабана и диска. Дополнительные подробности будут приведены в конце этого раздела (см. пример 8.7). Разумеется, также должен быть известен тип устройства, которое будет использовано (магнитные сердечники с разрушающим считыванием или полупроводники с неразрушающим считыванием).

В итеративной процедуре проектирования значение $\bar{v}_{цп}$, определенное в примере 8.1, можно использовать для оценки начальных значений параметров основной памяти. Эти значения будут затем представлять производительность основной памяти на итеративных этапах процедуры и могут быть изменены, если будет необходимо для удовлетворения требованиям проекта. Или же они будут основой для выбора из существующих вариантов основной памяти, параметры которой будут использованы (и, возможно, заменены на соответствующие значения для другой существующей памяти) в ходе итеративного анализа производительности и стоимости. В иерархическом методе проектирования коррекция первоначальных значений параметров основной памяти будет происходить по мере перехода к более низким уровням абстракции, т. е. по мере введения все более детальных сведений об архитектуре ЦП, о смеси команд, об интенсивностях обращений к вращающимся устройствам и их связях с основной памятью.

Важно отметить, что значение $\bar{v}_{цп}$ было вычислено по формуле (8.3) с определенным значением коэффициента h_N, который пропорционален стоимости, приходящейся на бит. Однако стоимость за бит связана с временем доступа, которое зависит от $\bar{v}_{цп}$. Таким образом, если из значения $\bar{v}_{цп}$ следует значение t_{a1}, которое не согласуется со стоимостью за бит, предполагаемой в нашей модели стоимости, то проектирование следует повторить, изменив подходящим образом h_N. Однако в случае первоначального проекта, аналогичного нашему, не имеет смысла проводить такую настройку на столь грубом уровне аппроксимации. Обычно полезнее перейти сразу же к итеративным этапам, на которых оперируют более детальной моделью системы и, начиная с некоторого момента, самой системой-прототипом.

Размеры диска и барабана следует определять одновременно с решением вопроса о распределении запоминаемой информации между ними и оценивая размеры распределяемых частей по отношению ко всей информации. При этом следует принять во внимание количество пользователей и их потребности, требования производительности и стоимости, а также предполагаемое расширение установки. Это исследование должно также дать оценки средних количеств обращений к диску и барабану на задание, т. е. $\bar{n}_{дс}$ и $\bar{n}_{бр}$. Как только выбраны размеры диска и барабана, можно определить коэффициенты h_2 и h_3 в модели стоимости (8.4). Для вращающихся устройств с неподвижными головками Шарп (1969) обнаружил, что их стоимость должна быть приблизительно пропорциональна корню квадратному из размера. Таким образом, h_2 в (8.4) будет пропорционально $\sqrt{M_2}$. Если M_2 и M_3 изменяются, система в принципе должна проектироваться заново по двум причинам: из-за изменения h_2 и h_3 и из-за изменения $\bar{n}_{бр}$ и $\bar{n}_{дс}$. ■

Пример 8.5. Большая часть обсуждения проекта иерархии памяти в примере 8.4 для ПРОП-системы пакетной обработки из примера 8.1 остается справедливой также

и для версии этой системы с виртуальной памятью, рассматриваемой в примере 8.2. Основные различия между этими двумя иерархиями памяти состоят в стратегии управления памятью (которая в данном случае является автоматической для иерархии основная память/барабан) и в функциях, отводимых вращающимся устройствам. В нашей системе с виртуальной памятью барабан используется только для страничного обмена. Это означает, что все задания, частично загруженные в основную память, должны сохранять целиком программу и данные (другими словами, свои пространства виртуальных адресов) на барабане, так чтобы любая не обнаруженная в памяти страница, на которую была ссылка, могла быть загружена с барабана. На диске будет храниться вся остальная информация, которая может понадобиться активным заданиям. Если информация, поступающая с диска, загружается в пространство виртуальных адресов задания, она может быть перенесена в процессе страничного обмена на барабан, если это требуется. В других случаях она будет переписана на диск или, если она более не нужна, уничтожена. В любом случае барабан должен быть достаточно велик, чтобы вместить пространства виртуальных адресов всех активных заданий. Поскольку виртуальное пространство оказывается заполненным очень редко, особенно в машинах с длинными виртуальными адресами, этот размер можно существенно уменьшить. Однако приведенные выше вычисления всегда указывают порядок величины требуемого пространства. Следовательно, значение h_2' в (8.8) должно считаться пропорциональным $\sqrt{N}$, так как оно пропорционально $\sqrt{M_2}$, а M_2 пропорционально N. Внесение корректировки в результаты, полученные в примере 8.2, оставим читателю в качестве упражнения (см. задачу 8.6).

Следует отметить, что значение $\bar{c}_{\text{ЦП}}$, определенное в примере 8.2, требует, по-видимому, слишком короткого времени доступа к основной памяти для той стоимости на бит, которая предполагалась при использовании формулы (8.8). Мы обсудим эту проблему в примере 8.6. ■

В примерах 8.4 и 8.5 мы обсуждали проектирование трехуровневых иерархий памяти. Однако в первом случае управление иерархией было полностью неавтоматическим, а во втором была включена автоматически управляемая двухуровневая иерархия. Многоуровневые иерархии, в которых автоматическое управление распространяется на все уровни, уже были введены в нескольких машинах и, вероятно, получат широкое распространение в будущем. Это следует из существующей длительное время традиции введения в вычислительные системы усовершенствований, освобождающих пользователя от непосредственного взаимодействия с аппаратурой. В число этих усовершенствований входят операционные системы и языки высокого уровня.

Теперь покажем, как можно оценить и спроектировать автоматическую многоуровневую иерархию памяти на примере трехуровневой иерархии такого типа. Эта иерархия будет получена добавлением еще одного автоматически управляемого уровня между ЦП и основной памятью в системе, рассмотренной в примере 8.2. Заметим, что подход, который будет описан, можно применять к другим иерархиям памяти, обладающим такими же структурой, организацией и методами управления.

Как и для всех иерархий памяти, ожидаемое преимущество от введения «*кэш-памяти*» (или *высокоскоростного буфера*) между ЦП и основной памятью состоит в улучшении соотношения между стоимостью и производительностью. Кэш-память позволяет быстрому ЦП обрабатывать команды с максимальной скоростью, тогда как в то же

самое время большая часть необходимого пространства основной памяти реализуется с помощью медленной, а следовательно, и более дешевой элементной базы. Мы увидим в примере 8.6, что добавление уровня кэш-памяти к системе, спроектированной нами в примере 8.2, будет весьма успешным, поскольку эта система требует намного более высокой скорости обработки ЦП, чем предполагаемая скорость основной памяти. В частности, отметим, что кэш-память занимает в иерархии памяти (рис. 8.7) уровень, удовлетворяющий определению основной памяти, т. е. уровень, с которым непосредственно связан ЦП. Однако на нашей терминологии и символике это никак не отразится (см. рис. 8.7).

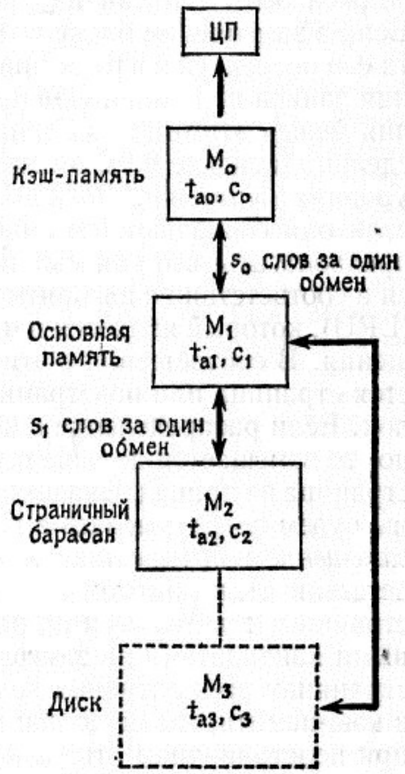


Рис. 8.7. Трехуровневая автоматически управляемая иерархия памяти, спроектированная в примере 8.6. Связь диска с уровнем основной памяти не управляется автоматически.

Существуют различные способы организации кэш-памяти. Мы обсудим один из них, наиболее согласующийся со способом управления остальными уровнями. Информация передается по запросу между основной памятью и кэш-памятью блоками, размер которых в общем случае является делителем размера страницы. Мы будем называть эти блоки *подстраницами* и делить кэш-память на *подстраничные области*. Отношение размера страницы s_1 к размеру подстраницы s_0 обычно равно степени числа 2. Если от ЦП поступает ссылка на слово, не находящееся в кэш-памяти, подстраница, содержащая это слово, передается из основной памяти в кэш-память. Если страница, содержащая подстраницу, на которую была ссылка, не находится в основной памяти, она должна быть извлечена из барабана и загружена в основную память до того, как желаемая подстраница будет передана в кэш-память. В некоторых системах передача запрошенной подстраницы в кэш-память может начинаться сразу же после того, как эта подстраница будет загружена в основную память. Аналогично, слово, на которое была ссылка, может быть передано в ЦП сразу же, как только оно появится в кэш-памяти. Мы, однако, сделаем более простое предположение, что пересылка должна быть завершена до того, как может быть инициирована следующая передача, включающая ту же информацию или ее часть.

Предполагается, что барабан содержит всю информацию, составляющую активное задание. Каждая подстраница в кэш-памяти порож-

дается некоторой страницей в основной памяти. Таким образом, в любой данный момент для любого активного задания информация, хранящаяся в кэш-памяти, является подмножеством информации, содержащейся в основной памяти, которая в свою очередь является подмножеством информации, содержащейся на барабане. Если пересылка невозможна без замены страницы или подстраницы, заменяемый блок должен быть записан на уровень, с которого поступает новый блок. Если в заменяемом блоке не было сделано никаких изменений с момента его последней загрузки на уровень, который он покидает, эта операция записи не нужна. Однако в дальнейшем мы не будем делать различия между этими двумя типами замещения (это различие *должно* быть сделано в задаче 8.9). Методы замещения могут отличаться на разных уровнях иерархии. Поскольку оценка иерархии памяти упрощается, если они совпадают [см., например, Маттсон и др. (1970)], мы будем предполагать, что как кэш-память, так и основная память управляются в соответствии с алгоритмом «самое давнее последнее использование» (LRU), который является наиболее распространенным алгоритмом замещения. В соответствии с этим алгоритмом, когда необходимо, замещается страница или подстраница, на которую дольше всего не было ссылок. Если распределение и кэш-памяти, и основной памяти фиксировано, то замещается дольше всех неиспользовавшаяся страница или подстраница из принадлежащих текущему заданию. Этот алгоритм, который мы будем подразумевать в нашем обсуждении, называется *локальным замещением дольше всех неиспользуемого*. С другой стороны, в соответствии с алгоритмом *глобального замещения* все страницы или подстраницы, хранящиеся на рассматриваемом уровне, являются возможными кандидатами на замещение [см. Деннинг (1970)]. Даже если подстраницам всех активных заданий разрешено одновременно находиться в кэш-памяти, мы не должны переключать ЦП на другое задание, если при подстраничном отказе нужная подстраница может быть извлечена из основной памяти. ЦП должен переключаться только в том случае, когда необходимая информация находится на барабане, т. е. если произошел страничный отказ (или, разумеется, если текущее задание выдает запрос на обмен с диском).

Пример 8.6. Процедура первоначального проектирования, обсуждавшаяся в примере 8.2, привела нас к системе со средней скоростью обработки в ЦП 1.4×10^6 команд/с. Повторяя рассуждения из примера 8.4 о задании начальных значений характеристик основной памяти, мы увидим, что при этой скорости, по-видимому, потребуется память со временем доступа в несколько сотен наносекунд. Допустим, $t_{a1}=300$ нс. Вспоминая, что первоначальный проект исходил из стоимости памяти около 1 цента за бит, мы можем убедиться из рис. 8.6, что мы недооценили эту стоимость. Пусть стоимость элементов с $t_{a1}=300$ нс составляет 4 цента за бит. Если в формуле (8.8) соответственно изменить значение h_M, кривые на рис. 8.4 изменятся. Например, новое решение, соответствующее минимальной стоимости среди изученных конфигураций, получается при значениях $N=2$ и $M=32K$ слов. Общая стоимость этого решения составит 132 600 долларов, а стоимость прошлого оптимального решения составит теперь 158 600 долларов. Заметим также, что, к сожалению, новое решение потребует даже большей величины $\bar{v}_{\text{цп}}$ (1.8×10^6 команд/с).

Мы хотим спроектировать трехуровневую иерархию памяти, включая кэш-память, так чтобы система удовлетворяла требованиям производительности, сформулированным в табл. 8.1. Наша цель состоит в том, чтобы уменьшить общую стоимость системы по сравнению с только что приведенной. Так как необходимо уточнить многие параметры проекта, мы не будем искать решения с минимальной стоимостью. Согласно нашей идеологии первоначального проектирования, мы выполним только несколько локальных оптимизаций, результаты которых можно эффективно использовать для уменьшения числа различных конфигураций, которые предстоит рассмотреть в ходе более точной, но все еще предварительной процедуры проектирования. Наш подход будет состоять в том, чтобы в качестве первоначального взять проект из примера 8.2 и добавить к нему кэш-память без изменения производительности. Время доступа кэш-памяти должно быть порядка 300 нс, так чтобы соответствовать $\bar{v}_{\text{цп}}$.

Легко понять, что если μ_2, N и $\bar{e}$ равны соответствующим величинам в решении примера 8.2 ($\mu_2=27.4\ \text{с}^{-1}$, $N=3$, $\bar{e}=45.5$ мс) и если спецификации рабочей нагрузки и производительности остаются прежними, то наша процедура проектирования даст те же значения характеристик остальных компонентов. Таким образом, мы заменим основную память двухуровневой системой памяти, которая при $N=3$ позволяет каждому заданию иметь средний интервал выполнения не менее 45.5 мс.

Из параметров автоматической иерархии, перечисленных на рис. 8.7, значения s_1, t_{a2}, c_2 и M_2 уже были выбраны (заметим, что $s_1=\bar{r}_{\text{бр}}=1K$ слов). Таким образом, еще не определенными параметрами проекта будут M_0, s_0, t_{a0}, M_1, t_{a1}, c_0 и c_1. Параметры t_{a0} и c_0, t_{a1} и c_1 связаны друг с другом, как показано на рис. 8.6. Первые пять из перечисленных параметров можно использовать для аналитического выражения эквивалентного времени доступа к иерархии памяти и среднего интервала выполнения $\bar{e}$, как описано ниже.

Для заданной программы и заданного алгоритма замещения кривая времени жизни обычно изменяется с изменением размера подстраниц на первом уровне иерархии. Таким образом, если бы не было кэш-памяти (а основная память обладала бы скоростью кэш-памяти), средний интервал выполнения был бы $\bar{e}(m_1, s_1)$, где $m_1=M_1/N$. Чтобы получить $\bar{e}$ при наличии кэш-памяти, мы должны добавить к $\bar{e}(m_1, s_1)$ время пересылки подстраницы s_0t_{a1}, умноженное на среднее число пересылок между кэш-памятью и основной памятью в промежутке между последовательными страничными отказами, равное $[\bar{e}(m_1, s_1)/\bar{e}(m_0, s_0)]-1$, где через $\bar{e}(m_0, s_0)$ обозначен средний интервал между подстраничными отказами (рис. 8.8). Используя $\bar{e}_i$ в качестве сокращенного обозначения для $\bar{e}(m_i, s_i)$, получим

$$\bar{e}=\bar{e}_1+\left(\frac{\bar{e}_1}{\bar{e}_0}-1\right)s_0t_{a1}. \tag{8.18}$$

Заметим, что мы записали время пересылки подстраницы в виде s_0t_{a1}, так как предполагаем, что уровень 1 представляет собой память произвольного доступа со скоростью пересылки $1/t_{a1}$. Если бы это была память на магнитных сердечниках с циклом $t_{\text{цикл}}$, то время пересылки было бы равно $s_0t_{\text{цикл}}$. Если бы это был, скажем, барабан, время пересылки было бы равно $\bar{t}_{a1}+s_0t_{\text{сбр}}=1/\mu_1$, где через μ_1 обозначена средняя интенсивность обслуживания барабаном.

Из рис. 8.8 мы можем видеть, что в течение одного цикла «ЦП — барабан» средняя программа производит $(\bar{e}_0/t_{a0})(\bar{e}_1/\bar{e}_0)=\bar{e}_1/t_{a0}$ обращений к иерархии памяти. Общая продолжительность этого цикла, включая пересылку подстраниц, следующую за пересылкой страниц, равна (в случае однопрограммной среды) $\bar{e}+1/\mu_2+s_0t_{a1}$. Таким образом, эквивалентное время доступа к иерархии для единственной программы, согласно (8.18), равно

$$t_{ae}=\frac{t_{a0}}{\bar{e}_1}\left(\bar{e}+\frac{1}{\mu_2}+s_0t_{a1}\right)=t_{a0}\left(1+\frac{s_0}{\bar{e}_0}t_{a1}+\frac{1}{\bar{e}_1\mu_2}\right). \tag{8.19}$$

Во время пересылки страницы ЦП может быть переключен на другие программы. Таким образом, это эквивалентное время доступа в течение интервалов, помеченных «ЦП» на рис. 8.8, которое должно соответствовать значению $\bar{v}_{\text{цп}}$, определенному в примере 8.2. Это время, обозначенное t^*_{ae}, не равно t_{ae}, но может быть легко получено из рассуждения, относящегося к (8.19):

$$t^*_{ae} = t_{a0}\left(1 + \frac{s_0}{\bar{e}_0}\, t_{a1}\right). \tag{8.20}$$

Для любой заданной программы можно построить семейство кривых времени жизни $\bar{e}(m, s)$ при различных значениях s (см., например, рис. 3.8.1). Если можно выделить одну или несколько программ, с достаточной вероятностью представляющих

Рис. 8.8. Временная диаграмма для автоматической трехуровневой иерархии рис. 8.7. Диаграмма относится к среднему заданию, выполняемому в однопрограммном режиме (очередей нет). Отрезки, выделенные жирной линией, представляют пересылки из основной памяти в кэш-память.

типичные применения проектируемой системы, эти кривые дадут необходимые данные для вычисления (8.18), (8.19) и (8.20) для разнообразия возможных конфигураций. Таким образом, итеративная процедура проектирования может использоваться для направленного исследования пространства параметров с целью получения удовлетворительного проекта. Аналитические формулы, которые могут помочь в этом поиске, можно вывести, введя простые математические описания семейства кривых времени жизни для средней программы. Следуя Желенбу (1973), мы предположим, что значение $\bar{e}(m, s)$ не зависит от s, и будем аппроксимировать его экспоненциальной функцией времени жизни $(a/\bar{v}_{\text{цп}})m^k$.

Стоимости системных компонентов, рассмотренные в примере 8.2, за исключением стоимости основной памяти, не изменятся, если $\bar{e}$ постоянно. Таким образом, нас интересует стоимость системы «кэш-память + основная память», которая задается выражением

$$C_{01} = c_0 M_0 + c_1 M_1. \tag{8.21}$$

Предположим, что выбраны все параметры проекта, кроме M_0 и M_1. Исследуя (8.20) и (8.21), читатель поймет, что при уменьшении M_0 и M_1 значение C_{01} уменьшается, а t^*_{ae} увеличивается. Поэтому мы будем искать локально оптимальное соотношение «стоимость — производительность», пытаясь минимизировать произведение $C_{01}t^*_{ae}$. Введем отношение $\alpha = M_1/M_0$ и выразим произведение $C_{01}t^*_{ae}$ как функцию от α. Из (8.20) и (8.21) получим

$$C_{01}t^*_{ae} = t_{a0}M_1\left(\frac{c_0}{\alpha} + c_1\right)\left(1 + \frac{s_0 t_{a1} N^k \bar{v}_{\text{цп}}}{aM_1^k}\,\alpha^k\right), \tag{8.22}$$

а из (8.18) —

$$M_1 = \left[\frac{N^k \bar{v}_{\text{цп}}}{a}\left(\bar{e} - s_0 t_{a1} - s_0 t_{a1}\alpha^k\right)\right]^{1/k}. \tag{8.23}$$

Подставляя (8.23) в (8.22), мы получим выражение для $C_{01}t^*_{ae}$, в котором единственной независимой переменной является α. Вычислив производную этого выражения и приравняв ее нулю, получим следующее уравнение:

$$\alpha^{k+1} + \frac{c_0}{c_1}\,\frac{k}{k-1}\,\alpha^k - \frac{c_0}{c_1(k-1)}\left(1 + \frac{\bar{e}}{s_0 t_{a1}}\right) = 0. \tag{8.24}$$

Заметим, что это уравнение не содержит ни a, ни t_{a0}, ни N. Если мы выберем $k=2$, $c_0/c_1=5$, $s_0=64$ словам, а $t_{a1}=1$ мкс, решение будет $\alpha \approx 12.5$. Из (8.23) мы получим $M_1=57\,849$ словам, а $M_0=M_1/\alpha=4627$ словам. Предполагая $t_{a0}=0.25$ мкс, получим из (8.20) $t^*_{ae}=320$ нс. Стоимость, вычисленная по формуле (8.21) при $c_1=1$ цент/бит и $c_0=5$ цент/бит, равна приблизительно 25 900 долларам. Если мы выберем более разумные размеры для основной памяти и кэш-памяти, например $M_0=4K$ слов и $M_1=64K$ слов, приведенные выше значения станут равными $t^*_{ae}=340$ нс и $C_{01}=27\,500$ долларам. Таким образом, общая стоимость системы составит всего 106 100 долларов вместо 158 600 долларов, вычисленных в начале этого примера для двухуровневой иерархии. Заметим, что (8.18) дает $\bar{e}=55.4$ мс вместо 45.5 мс. Таким образом, благодаря нашему новому выбору M_0 и M_1 мы получили также небольшое увеличение производительности. ■

При проектировании иерархии, которое мы провели в примере 8.6, процедура распадалась на два последовательных этапа. Первый этап (описанный в примере 8.2) не предполагал наличия кэш-памяти и дал значения характеристик всех компонентов системы. Стоимость получившейся в результате системы оказалась, однако, довольно большой вследствие необходимости использования основной памяти с высокой скоростью. Затем на втором этапе основная память была заменена на двухуровневую иерархию без изменения остальных компонентов, так что производительность системы не изменилась. При таком подходе, который можно рассматривать как частный случай иерархического проектирования (см. разд. 8.2 и 8.4), общая стоимость системы существенно уменьшилась. При проектировании системы этого типа можно следовать и некоторым другим подходам. Например, можно начать с конфигурации, уже содержащей трехуровневую иерархию, задать разумные значения параметров кэш-памяти и основной памяти, определить $\bar{e}$ для различных значений N с помощью (8.18) или подходящих кривых времени жизни $\bar{e}(m, s_0)$ и $\bar{e}(m, s_1)$ и продолжать проектирование, как показано в примере 8.2, убедившись, что результирующее значение $\bar{v}_{\text{цп}}$ соответствует значению t^*_{ae}, полученному из (8.20). Другое возможное использование того же основного набора аналитических и эмпирических средств описано Трейгером и Маттсоном (1972), которые выбирали значения всех параметров системы, кроме M_0, M_1, s_0, s_1. Для каждой элементной базы основной памяти, характеризуемой парой чисел (t_{a1}, c_1), они определяли значения M_0, M_1, s_0, s_1, оптимизирующие индекс «стоимость — производительность», общую арендную плату на одну выполняемую команду. Сравнение значений этого индекса, полученных для различных элементных баз памяти, позволяет проектировщикам системы выбрать элементную базу, наиболее подходящую для их системы, и оценивать новые элементные базы.

Более того, проектировщики памяти могут применить тот же подход для определения наиболее эффективных способов усовершенствования определенного типа памяти, например будет ли лучше понизить стоимость элемента памяти или увеличить его быстродействие.

До сих пор мы оценивали необходимую скорость основной памяти довольно грубым способом (см. примеры 8.4 и 8.6). Более детальные вычисления должны учитывать влияние различных устройств, непосредственно обращающихся к основной памяти. Теперь мы кратко обсудим эти воздействия для двух систем, проектирование которых рассмотрено в примерах 8.1 и 8.6. Результаты этих вычислений явятся отправным пунктом для детального проектирования основной памяти, которого мы не будем касаться в данной книге.

Рис. 8.9. Блок-схема системы, спроектированной в примере 8.1.

Пример 8.7. На рис. 8.9 изображена блок-схема системы, проектирование которой рассматривалось в примере 8.1. Блок-схема отражает принятую здесь точку зрения, согласно которой основная память рассматривается как центр системы и может быть доступна нескольким процессорам через каналы прямого доступа (DMA). В нашем примере есть три процессора: ЦП, контроллер барабана (или канал барабана), контроллер диска (или канал диска). Мы, разумеется, игнорируем процессор (или процессоры) ввода-вывода, обеспечивающий передачу данных по каналу ввода-вывода от устройства ввода перфокарт или другого входного устройства к диску и от диска к печатающему устройству или другому выходному устройству. Во время этих пересылок данные обычно накапливаются в буфере основной памяти. Игнорирование интенсивности запросов ввода-вывода законно, если, как это часто бывает, она на несколько порядков ниже, чем для остальных процессоров.

Какова максимальная интенсивности обращений к памяти для системы, представленной на рис. 8.9? Чтобы ответить на этот вопрос, мы должны вычислить интенсивности обращений этих трех процессоров к памяти и сложить их. ЦП должен обрабатывать $\bar{v}_{цп}$ команд/с. Сколько обращений к памяти они потребуют? Рассмотрим смесь команд, удовлетворительно представляющую состав программы, которую должна обрабатывать наша система. Если команда i имеет относительную частоту f_i и требует g_i обращений к основной памяти, то средняя интенсивность обращений ЦП к памяти равна

$$\bar{f}_{цп} = \bar{v}_{цп} \sum_i f_i g_i. \tag{8.25}$$

В табл. 8.3 собрана информация, необходимая для вычисления $\bar{f}_{цп}$. Эта информация может быть получена из обычной смеси команд очевидным способом. Полагая $\bar{v}_{цп} = 1.26 \times 10^6$ команд/с, как было определено в примере 8.1, получим из (8.25) $\bar{f}_{цп} = 2.4 \times 10^6$ обращений/с. Заметим, что этот подход нельзя использовать для конвейерного ЦП [1]). В этом случае требуется имитационное моделирование или более сложный анализ.

Процессор барабана при передаче данных обращается к памяти с интенсивностью

$$f_{бараб} = \frac{1}{t_{сбр}}, \tag{8.26}$$

[1]) Конвейерный процессор обеспечивает параллельное выполнение микроопераций во всех разрядах одновременно.— *Прим. ред.*

и дисковый процессор — с интенсивностью

$$f_{диск} = \frac{1}{t_{сдс}}. \tag{8.27}$$

Интенсивности обращений диска и барабана к памяти постоянны, тогда как интенсивность обращений от ЦП меняется со временем. Для нашей системы $f_{бараб} = 10^5$ обращений/с и $f_{диск} = 0.77 \times 10^5$ обращений/с. Общая интенсивность обращений к памяти при работе всех трех процессоров равна

$$f_{общ} = \bar{f}_{цп} + f_{бараб} + f_{диск}. \tag{8.28}$$

В нашей системе $f_{общ} = 2.577 \times 10^6$ обращений/с. Когда процессоры барабана или диска обращаются к памяти, они обладают более высоким приоритетом, чем ЦП, который следовательно, будет работать медленнее в случае возникновения конфликта из-за доступа к памяти между ним и любым из остальных процессоров. Если мы хотим, чтобы нашу память можно было использовать при интенсивности $f_{общ}$ без того, чтобы на протяжении относительно больших интервалов величина $\bar{v}_{цп}$ оказалась существенно ниже вычисленного нами значения, мы должны сделать скорость пересылки у памяти, или ее *полосу пропускания*, большей или равной $f_{общ}$. Разумеется, это в некотором смысле пессимистическая точка зрения, так как мы предполагаем, что все процессоры будут конкурировать за доступ к памяти. С другой стороны, мы делаем оптимистическое предположение, что ЦП способен использовать все циклы памяти. Если выбраны элементы на магнитных сердечниках, продолжительность цикла по крайней мере должна быть равна $1/f_{общ} = 388$ нс. Изготовление памяти на магнитных сердечниках в виде единственного модуля с полосой пропускания, соответствующей времени доступа 200 нс, довольно дорого. Поэтому для получения требуемой полосы пропускания при использовании сравнительно медленных элементов необходимо разделение на модули. Кроме того, для дальнейшего увеличения скорости многомодульной памяти можно использовать такие хорошо известные методы, как, например, чередование обращений и сдвиг циклов. Расширение пути доступа может увеличить производительность как одномодульных, так и многомодульных блоков памяти [описание этих методов см., например, у Уотсона (1970)]. В случае полупроводниковых элементов время цикла приблизительно совпадает с временем доступа, а времена доступа около 400 нс не очень трудно или дорого получить. Впрочем, названные выше методы можно использовать для расширения полосы пропускания памяти и этого типа.

Таблица 8.3

Смесь команд для примера 8.7

Число обращений на одну команду	Относительная частота (%)
1	22
2	67
3	9
4	2

В системе с виртуальной памятью, проектирование которой рассматривалось в примере 8.6, ЦП не имеет прямой связи с основной памятью. ЦП на рис. 8.9 следует заменить кэш-памятью, которая характеризуется при передаче данных в основную память или из нее постоянной интенсивностью обращений $f_{кэш} = 1/t_{a0}$. Если значение t_{a0} равно 250 нс, как в примере 8.6, то $f_{кэш} = 4 \times 10^6$ обращений/с. Чтобы соответствовать этой интенсивности обращений, полоса пропускания основной памяти по отношению к кэш-памяти должна быть расширена в 4 раза (так как $t_{a1} = 1$ мкс).

Расширение пути доступа посредством передачи 4 слов одновременно является распространенным методом достижения этого результата. Чтобы удовлетворить требованиям ЦП, максимальная скорость передачи данных у кэш-памяти должна быть не меньше $\bar{f}_{цп}$. В нашей системе с виртуальной памятью, предполагая, что использована смесь команд из табл. 8.3, и взяв $\bar{v}_{цп}=1.4\times10^6$ команд/с, как диктуется процедурой проектирования в примере 8.2, мы получим из (8.25) $\bar{f}_{цп}=2.67\times10^6$ обращений/с и максимальное значение t_{a0}, равное $1/\bar{f}_{цп}=374$ нс. ■

8.3.3. Исследование компромиссов проектирования

Проектировщик обычно встречается с целым рядом компромиссов, каждый из которых заключает в себе выбор между противоречивыми требованиями стоимости и ценности. Пусть рассматривается с точки зрения внедрения в проект некоторая особенность системы, предположительно повышающая ценность этой системы для пользователей. Оправдывает ли ожидаемое увеличение ценности затраты на эту особенность и неудобства, обусловленные возможными побочными эффектами? Несколько примеров компромиссов между стоимостью и производительностью, а также между различными компонентами оборудования было приведено в гл. 7 и в разд. 8.3.1 и 8.3.2. Примером компромисса между различными компонентами оборудования служит компромисс между размером основной памяти и скоростью страничного барабана в системе с виртуальной памятью. Другие компромиссы, которые предстоит рассматривать при проектировании системы, это компромиссы между аппаратурой и программным обеспечением. Последние возникают естественным образом, когда приходится проектировать системные компоненты или когда среди имеющихся компонентов оборудования некоторые выполняют определенные функции, а некоторые нет (так что, если выбраны последние, эти функции должны выполняться программным обеспечением).

В прошлом стоимость и негибкость оборудования заметно ограничивали свободу проектировщика. Оптимальная граница между областями оборудования и программного обеспечения постоянно перемещалась вследствие развития технологии, но область альтернативных решений вблизи границы была относительно узкой. Появление микропрограммирования и крупномасштабная интеграция открыли новые перспективы и заметно расширили возможности выбора для проектировщика. Появилась возможность изготавливать части операционной системы в микропрограммном виде и даже в виде аппаратуры (или в виде комбинации аппаратуры с микропрограммным обеспечением). Какая же из возможных альтернатив является наилучшей? Какая окажется наилучшей через год?

Ответ на эти вопросы требует исследований не только по оценке производительности. Необходимо тщательно изучить экономические и функциональные факторы наряду с неколичественными воздействиями на установки и коллективы их пользователей. Однако очевидно, что проблема производительности является важной стороной компромиссов между аппаратурой и программным обеспечением. Исследование, выполненное Фоули (1971), по оптимальному проектированию системы машинной графики будет описано в качестве примера того, как можно оценить аспекты критерия стоимость — производительность для компромиссов «аппаратура — аппаратура» и «аппаратура — программное обеспечение».

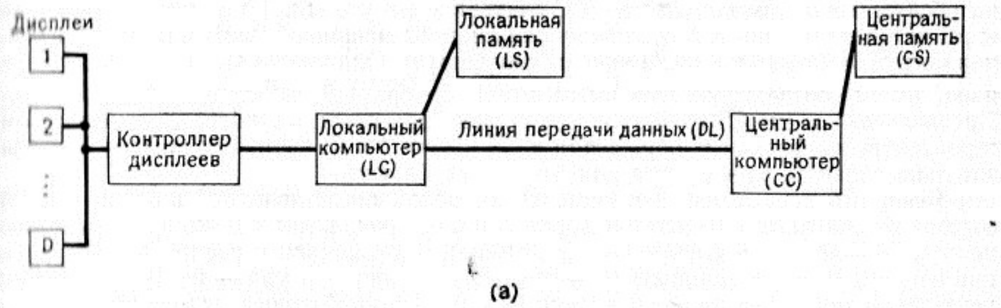


Рис. 8.10. Система машинной графики (a) и модель (b), рассмотренные в работе Фоули (1971). Блоки в (b) представляют различные функции, выполняемые соответствующими физическими устройствами в (a). Приоритеты этих функций, если необходимо, указаны в скобках внутри каждого блока.

Пример 8.8. Конфигурация проектируемой диалоговой системы машинной графики показана на рис. 8.10(a). Модель этой системы изображена на рис. 8.10(b). Эта модель описывает возможные пути прохождения запросов пользователя через систему. Запрос пользователя может предназначаться для обслуживания центральным компьютером или для локального обслуживания. В первом случае запрос должен быть предварительно обработан локальным компьютером LC [блок 6 на рис. 8.10 (b)], переслан по линии передачи данных DL (блок 7) и обработан центральным компьютером CC (блок 8), возможно, с обращениями к основной памяти центрального компьютера CS (блок 9). Затем результаты будут переданы обратно по линии DL (блок 10), дополнительно обработаны в локальном компьютере LC (блок 11) и, наконец, будут выданы пользователю. Команда, требующая локального обслуживания, обрабатывается в LC (блок 1). Иногда такая команда может включать одно или несколько обращений к локальной памяти LS (блок 2) или к центральной памяти CS (блоки 3,4 и 5). Блоки модели представляют отдельные центры обслуживания, которые иногда реализуются тем же физическим устройством. В этих случаях устройство будет обслуживать

запросы в соответствии с фиксированными приоритетами, изображенными в скобках внутри соответствующих блоков на рис. 8.10(b).

В структуре модели неявно присутствует ряд упрощающих предположений. Отсутствие стрелки обратной связи вокруг блока 6 означает, что компьютер LC должен завершить предварительную обработку до того, как соответствующие сообщения будут переданы по линии DL. Кроме того, LC не может обращаться к LS во время предварительной или завершающей обработки запроса. При обращении локального компьютера LC (блок 4) к CS для удовлетворения локально обрабатываемого запроса на обслуживание вмешательство CC считается ненужным. Компьютер CC, разумеется, не предназначен для обслуживания системы машинной графики, дополнительная нагрузка на него в модели явно не фигурирует. Она вместо этого учитывается неявно, путем соответствующего уменьшения его средней скорости выполнения $\bar{v}_{CC}$. Предполагается, что CS состоит из нескольких дисковых устройств. Таким образом, вероятность конфликтных обращений к CS (блоки 4 и 9) настолько мала, что необязательно задавать приоритеты для этих двух блоков.

Ясно, что проектирование системы, подобной представленной на рис. 8.10(a), состоит не только в определении характеристик программных и аппаратных компонентов. Необходимо исследовать ряд проблем, относящихся к распределению функций и компромиссам «аппаратура — аппаратура», «аппаратура — программное обеспечение». К типичным проблемам распределения функций относится проблема выбора, какие задачи должны обрабатываться в локальном компьютере, а какие — в центральном. Примерами компромиссов «оборудование — оборудование» служит разделение вычислительной мощности между центральным и локальным процессорами, а также определение размеров локальной и центральной основной памяти. Примером компромисса «оборудование — программное обеспечение» является возможная реализация ряда функций вывода контроллером дисплея или их выполнение программами, работающими в локальном компьютере. Аппаратная реализация этих функций обычно дает меньшее время выполнения, но оказывается дороже и труднее поддается изменению, чем программная реализация.

Для сравнения возможных решений проблемы проектирования будет выбран критерий стоимость — производительность. В качестве оценки, которая будет максимизироваться, возьмем среднюю пропускную способность на единицу стоимости, измеряемую во взаимодействиях в секунду на доллар:

$$\frac{\bar{T}}{C} = \frac{D}{\bar{t} + \bar{t}_t} \cdot \frac{1}{C}, \qquad (8.29)$$

где D — число дисплеев, обслуживаемых системой, $\bar{t}$ — среднее время ответа, $\bar{t}_t$ — среднее время обдумывания, а C — общая стоимость. Так как значение $\bar{t}_t$ задано (см. табл. 8.4), D известно, а C можно оценить на основании стоимостей компонентов, то единственная неизвестная величина, которую предстоит вычислить по (8.29), это $\bar{t}$. Модель в виде замкнутой сети с очередями на рис. 8.10(b) позволяет нам определить $\bar{t}$ аналитически, если предположить что вероятности ветвления постоянны, а времена обслуживания во всех обслуживающих центрах распределены экспоненциально. Имитационная модель, в которой эти предположения не были выполнены, дала результаты, отличающиеся на множестве проверенных ситуаций не более чем на 7% от полученных аналитически при тех же средних значениях как вероятностей ветвления, так и времен обслуживания.

В табл. 8.4 перечислены те параметры оборудования и прикладных программ, на основании которых должны вычисляться вероятности ветвления и средние времена обслуживания. В выборе и определении этих параметров явно или неявно присутствует ряд допущений. Единицей пространства памяти является *запись*, равная 12 000 бит. Все дисплейные прикладные программы считаются состоящими из *макрокоманд обслуживания дисплея*. Эти макрокоманды можно в некотором смысле рассматривать как единицы рабочей нагрузки, которую предстоит обрабатывать. Типичными примерами таких макрокоманд являются инициализация дисплея, поворот, мерцание линии, изменение световой интенсивности, перевод целого числа в код ASCII,

получение текущих координат и т. д. *Прикладные программы работы с дисплеем* подразделяются на различные типы, например редактирование текстов, вывод двумерных изображений, анализ сетей. Каждый тип прикладных программ характеризуется последовательностью *взаимодействий* (или *запросов на обслуживание*). Например, прикладная программа анализа сетей будет включать такие взаимодействия: передвинуть элемент, идентифицировать элемент с помощью светового пера, удалить

Таблица 8.4

Параметры оборудования и прикладных программ для графической системы рис. 8.10(a)

Обозначение	Определение
$\bar{t}_t$	Среднее время обдумывания для пользователя (с)
$\bar{v}_{LC}$	Средняя скорость выполнения макрокоманд дисплея локальным компьютером (макрокоманды/с)
M_{LC}	Размер основной памяти локального компьютера (в записях)
$\bar{t}_{LS}$	Среднее время обслуживания в локальной памяти (с/запись)
M_{LS}	Размер локальной памяти (в записях)
v_{DL}	Скорость передачи в линии передачи данных (бит/с)
$\bar{v}_{CC}$	Средняя скорость выполнения макрокоманд дисплея центральным компьютером (макрокоманд/с)
$\bar{t}_{CS}$	Среднее время обслуживания в центральной памяти (с/запись)
G	Число типов запросов обслуживания
L_i	Число макрокоманд дисплея, необходимых для обслуживания запроса типа i ($i=1,2,\ldots,G$)
p_i	Вероятность обслуживания запроса типа i ($i=1,2,\ldots,G$)
n_i^{BS}	Число обращений к памяти, необходимых для обработки запроса типа i в локальном компьютере, если на программу приходится одна запись в основной памяти ($i=1,\ldots,G$)
n_i^{CS}	Число обращений к центральной памяти, необходимых для обработки запроса типа i в центральном компьютере ($i=1,2,\ldots,G$)
L_{pp}	Число макрокоманд дисплея, которые должны быть выполнены локальным компьютером для предварительной или завершающей обработки (блоки 6 и 11)
F	Максимальный размер прикладной программы и базы данных (в записях)
a_j	Вероятность обращения к записи j файла ($j=1,2,\ldots,F$); записи в файле пронумерованы в порядке убывания вероятности обращения
E	Минимальный размер памяти локального компьютера, необходимый для работы дисплейной прикладной программы (записей)
L_m	Длина сообщения, передаваемого по линии передачи данных (бит)

Источник: Фоули (1971).

идентифицированный элемент, связать идентифицированные элементы, удалить связь и т. д. Каждое взаимодействие в свою очередь характеризуется числом L_i макрокоманд вывода, необходимых для его выполнения, своей вероятностью p_i, числом обращений n_i^{BS} к основной памяти (LS или CS или к обеим), которое требуется ему для обработки в LC, и числом n_i^{CS} обращений к CS при обработке в CC. Числа

макрокоманд вывода для каждого взаимодействия достаточно, чтобы представить его вычислительные потребности, так как предполагается, что для каждой прикладной программы задана смесь макрокоманд. Средняя скорость выполнения макрокоманд может быть вычислена по смеси [см. (5.9)] как для LC ($\bar{v}_{LC}$), так и для CC ($\bar{v}_{CC}$). Как уже было сказано выше, скорость $\bar{v}_{CC}$ нужно уменьшить, чтобы учесть воздействие дополнительной нагрузки на CC.

Максимальный размер информации, необходимой прикладной программе, равен F записям. Записи файлов прикладных программ пронумерованы в порядке убывания вероятности обращения a_j. Таким образом, если E — минимальное количество основной памяти в LC, посредством которого могут быть реализованы функции дисплея, не обеспечиваемые аппаратно, но требующиеся для данной прикладной программы, то первые $M_{LC}-E$ записей файла будут находиться в основной памяти, следующие M_{LS} будут находиться в LS, а остальные $F-M_{LS}-M_{LC}+E$ записей должны будут находиться в CS. Заметим, что мы предполагаем, что все терминалы работают с одной прикладной программой. Большинство параметров в табл. 8.4 изменяется при переходе к другой прикладной программе. Таким образом, при

$$A(k)=\sum_{j=1}^{k} a_j \quad (k=1, \ldots, F) \tag{8.30}$$

вероятности нахождения необходимой записи в основной памяти LC, LS и CS соответственно равны

$$\begin{aligned} p_{LC} &= A(M_{LC}-E), \\ p_{LS} &= A(M_{LS}+M_{LC}-E)-p_{LC}, \\ p_{CS} &= 1-p_{LC}-p_{LS}. \end{aligned} \tag{8.31}$$

Задачи исполнения прикладной программы будут распределены между LC и CC в соответствии с критерием минимального времени ответа. Пусть Λ — набор индексов для взаимодействий, относящихся к LC, а Γ — для взаимодействий, относящихся к CC. Таким образом, вероятность $b_{0,1}$ того, что пользователь выдал запрос на локальное обслуживание, равна

$$b_{0,1}=\sum_{i\in\Lambda} p_i. \tag{8.32}$$

Вероятность запроса на обслуживание в CC равна

$$b_{0,6}=1-b_{0,1}=\sum_{i\in\Gamma} p_i. \tag{8.33}$$

Среднее число обращений к основной памяти на одно взаимодействие, поступивших от LC, если он используется и если только одна запись в памяти LC доступна как область программы, равно

$$\bar{n}^{BS}=\sum_{i\in\Lambda}\frac{p_i}{b_{0,1}} n_i^{BS}, \tag{8.34}$$

а число обращений, поступивших от CC, если он используется, равно

$$\bar{n}^{CS}=\sum_{i\in\Gamma}\frac{p_i}{b_{0,6}} n_i^{CS}. \tag{8.35}$$

Если $M_{LC}-E$ записей в памяти доступны для LC, среднее число обращений к основной памяти (либо LS, либо CS) окажется приблизительно равным $(1-p_{LC})\bar{n}^{BS}$. Среднее число обращений к блоку 1 равно $1+(1-p_{LC})\bar{n}^{BS}$, а за последним обращением следует возврат в блок 0. Поэтому

$$b_{1,0}=\frac{1}{1+(1-p_{LC})\bar{n}^{BS}}, \tag{8.36}$$

и аналогично

$$b_{1,2}=\frac{p_{LS}\bar{n}^{BS}}{1+(1-p_{LC})\bar{n}^{BS}}, \tag{8.37}$$

$$b_{1,3}=\frac{p_{CS}\bar{n}^{BS}}{1+(1-p_{LC})\bar{n}^{BS}}=1-b_{1,0}-b_{1,2}. \tag{8.38}$$

Среднее время, необходимое для взаимодействия с запросом на локальное обслуживание, имеет вид

$$\bar{t}_{1\,общ}=\sum_{i\in\Lambda}\frac{p_i}{b_{0,1}}\frac{L_i}{\bar{v}_{LC}}, \tag{8.39}$$

а для взаимодействия с участием CC —

$$\bar{t}_{8\,общ}=\sum_{i\in\Gamma}\frac{p_i}{b_{0,6}}\frac{L_i}{\bar{v}_{CC}}. \tag{8.40}$$

Так как число прохождений через обслуживающий прибор 1 при локальном взаимодействии равно $1/b_{1,0}$, а через обслуживающий прибор 8 аналогично — $1/b_{8,10}$ при центральном взаимодействии, то средние времена обслуживания $\bar{t}_1$ и $\bar{t}_8$ можно вычислять по формулам табл. 8.5. Эта таблица содержит уравнения, выражающие вероятности ветвлений и средние времена обслуживания для модели с рис. 8.10 (b) как функции параметров, приведенных в табл. 8.4 или уже полученных нами из них.

Таблица 8.5

Вероятности ветвлений и средние времена обслуживания для модели рис. 8.10 (b)

$b_{0,1}=\sum_{i\in\Lambda} p_i$ [1]	$\bar{t}=b_{1,0}\bar{t}_{1\,общ}$ [4]
$b_{0,6}=1-b_{0,1}$	$\bar{t}_2=\bar{t}_{LS}$
$b_{1,0}=\frac{1}{1+(1-p_{LC})\bar{n}^{BS}}$ [2]	$\bar{t}_3=\bar{t}_5=\bar{t}_7=\bar{t}_{10}=\frac{L_m}{\bar{v}_{DL}}$
$b_{1,2}=\frac{p_{LS}\bar{n}^{BS}}{1+(1-p_{LC})\bar{n}^{BS}}$ [2]	$\bar{t}_4=\bar{t}_9=\bar{t}_{CS}$
$b_{1,3}=1-b_{1,0}-b_{1,2}$	$\bar{t}_6=\bar{t}_{11}=\frac{L_{pp}}{\bar{v}_{LC}}$
$b_{8,9}=\frac{\bar{n}^{CS}}{\bar{n}^{CS}+1}$ [3]	$\bar{t}_8=b_{8,10}\bar{t}_{8\,общ}$ [5]
$b_{8,10}=1-b_{8,9}$	

Источник: Фоули (1971).
[1]) См. (8.32).
[2]) См. (8.31) и (8.34).
[3]) См. (8.35).
[4]) См. (8.39).
[5]) См. (8.40).

Среднее время $\bar{t}'_h$, в течение которого запрос на обслуживание находится в обслуживающем центре h, каждый раз, когда он проходит через этот центр, будет равно

$$\bar{t}'_h = \bar{t}_h + \bar{t}_{wh} \quad (h = 1, 2, \ldots, 11), \tag{8.41}$$

где $\bar{t}_{wh}$ — среднее время ожидания в очереди центра. Времена $\bar{t}'_h$ можно вычислить аналитическими методами. Для вычисления среднего времени ответа заметим, что среднее время, проводимое запросом в цикле [см. рис. 8.10(b)] у обслуживающих центров 8 и 9, равно

$$\bar{t}'_{89} = \frac{\bar{t}'_8}{b_{8,10}} + \left(\frac{1}{b_{8,10}} - 1\right)\bar{t}'_9 = \frac{\bar{t}'_8 + b_{8,9}\bar{t}'_9}{b_{8,10}}, \tag{8.42}$$

так как в среднем запрос проходит $1/b_{8,10}$ раз через блок 8 и на один раз меньше через блок 9, прежде чем покинуть их. Применив тот же подход к циклу, составленному из блоков 1, 2 и (3, 4, 5), мы можем написать

$$\bar{t} = b_{0,1}\frac{\bar{t}'_1 + b_{1,2}\bar{t}'_2 + b_{1,3}(\bar{t}'_3 + \bar{t}'_4 + \bar{t}'_5)}{b_{1,0}} + b_{0,6}\left(\bar{t}'_6 + \bar{t}'_7 + \frac{\bar{t}'_8 + b_{8,9}\bar{t}'_9}{b_{8,10}} + \bar{t}'_{10} + \bar{t}'_{11}\right). \tag{8.43}$$

Заметим, что число запросов в сети с очередями, изображенной на рис. 8.10(b), равно в любой заданный момент числу активных терминалов, которое не может превышать D. Если, в частности, имеется только один активный терминал, то $\bar{t}_{wh} = 0$ для всех h в (8.41) и для вычисления $\bar{t}$ не требуется никакого анализа очередей, так как все переменные в (8.43) можно получить из уравнений табл. 8.5.

Как отражаются на только что описанном подходе компромиссы проектирования и проблемы распределения функций? То, что было названо компромиссами «аппаратура — аппаратура», очевидно, включается в выбор параметров компонентов оборудования в табл. 8.4. Например, мы можем заменить M_{LS} на v_{DL}. Иначе говоря, увеличение M_{LS} могло бы позволить нам уменьшить v_{DL} так, чтобы сохранить стоимость постоянной и повысить производительность по отношению к определенному проекту. Компромиссы «оборудование — оборудование» представлены по большей части E, пространством памяти, необходимым в LC для введения в программное обеспечение тех функций обслуживания терминала, которые не встроены в аппаратуру, а также временами выполнения тех макрокоманд обслуживания терминала, которые определяют скорости обработки $\bar{v}_{LC}$ и $\bar{v}_{CC}$. Из (8.39), (8.40) и табл. 8.5 можно видеть, как влияют большие значения этих величин на времена обслуживания. Кроме того, коль скоро значение E будет меньше, чем оно было бы, если бы пришлось встроить эти функции в программное обеспечение, то p_{LC} окажется больше, а p_{LS} и p_{CS} будут меньше [см. (8.31)]. Это приведет к увеличению $b_{1,0}$ [см. (8.36)], тогда как $b_{1,2}$ и $b_{1,3}$ уменьшатся [см. (8.37) и (8.38)]. Из (8.43) очевидно, что $\bar{t}$ будет уменьшаться в результате реализации некоторых макрокоманд аппаратными средствами, а стоимость, вообще говоря, возрастет. Альтернативное решение состояло бы в замене имеющегося оборудования на другое. Вместо включения дополнительного оборудования в состав терминального контроллера, можно было бы увеличить размер основной памяти LC так, чтобы разность $M_{LC}-E$ не изменилась. Это повлияло бы не на $b_{1,0}$, $b_{1,2}$ и $b_{1,3}$, а только на $\bar{v}_{LC}$ и $\bar{v}_{CC}$, что привело бы к возрастанию $\bar{t}$. Однако стоимость этих двух решений может заметно отличаться, и это отличие, вероятно, оказалось бы в пользу воплощения в программном обеспечении даже в этом случае.

Распределение функций влияет на состав множеств Λ и Γ. Следовательно, согласно (8.32) и (8.33), оно воздействует на $b_{0,1}$ и $b_{0,6}$, а согласно (8.34) и (8.35),— на $\bar{n}^{BS}$ и $\bar{n}^{CS}$. Из табл. 8.5 видно, что все вероятности ветвления $\bar{t}_1$ и $\bar{t}_8$ действительно зависят от распределения функций. Следовательно, можно ожидать, что значение $\bar{t}$ будет заметно изменяться в зависимости от принятого нами решения по этому вопросу. Мы будем принимать эти решения с целью минимизации $\bar{t}$, так как в предположениях работы Фоули они не оказывают заметного влияния на общую стоимость.

Компромиссы, которые мы уже обсудили, можно использовать для минимизации $\bar{T}/C$. Подход, которому следовал Фоули, состоит в рассмотрении конфигураций, которые получаются при комбинировании имеющихся компонентов всеми допустимыми способами. В частности, если среди компонентов на рис. 8.10(a) CC и CS предполагаются заданными, некоторая конфигурация получается выбором одной из возможных пар LC/контроллер дисплея, одного из возможных размеров основной памяти для LC, одного из возможных размеров для LS (предполагая одинаковую скорость для всех реализаций LS) и одной из возможных скоростей передачи для DL. Фоули описал базу данных, содержащую данные существующих компонентов, которая позволяет получить 12 480 различных конфигураций. Для нахождения оптимальных решений при различных значениях D, различных месячных затратах и различных типах дисплейных прикладных программ была

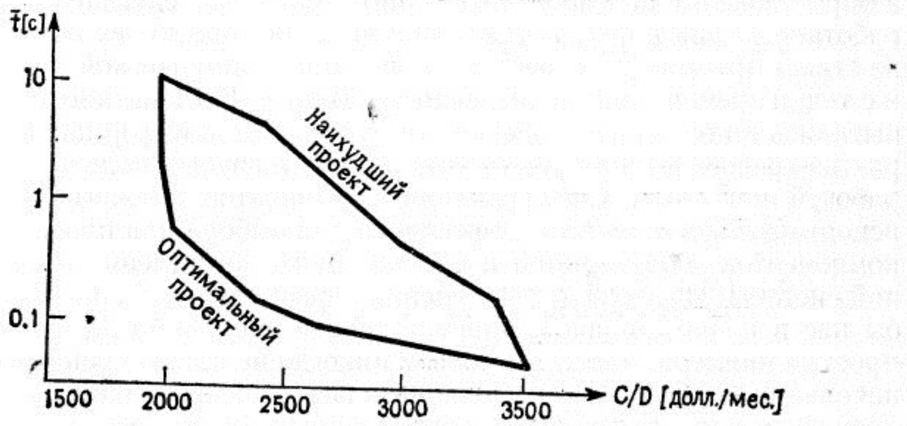


Рис. 8.11. Среднее время ответа как функция ежемесячных затрат на терминал для оптимального и наихудшего проекта. [*Источник:* Фоули (1971), с разрешения ACM.]

использована итеративная процедура оптимизации, которая осуществляла направленный поиск в пространстве возможных решений так, чтобы избежать проверки большей части конфигураций. Значения параметров табл. 8.4, которые следовало бы измерить, вместо этого были оценены, так как измеренных данных в наличии не было. Читатель должен учесть, что, несмотря на многочисленные и довольно сильные предположения, сделанные для упрощения всей модели и, в частности, модели рабочей нагрузки, может оказаться весьма трудно найти или определить значения параметров рабочей нагрузки, например L_i, p_i, n_i^{BS}, n_i^{CS}, L_{pp}, F и a_j, и относительные частоты макрокоманд в прикладных смесях. Только детальное знание прикладной программы может позволить проектировщику сделать разумные оценки этих параметров. Таким образом, мы видим, что проблема описания рабочей нагрузки не может быть решена полностью, пока не разработаны методы для надежной оценки или измерения параметров рабочей нагрузки или пока не будут предприняты обширные измерения рабочей нагрузки, результаты которых можно с достаточной степенью точности применить к различным типам приложений.

Некоторые из результатов Фоули представлены на рис. 8.11, показывающем преимущества оптимальных проектов по критерию стоимость — производительность. В соответствии с этой диаграммой при затратах на дисплей 2500 долларов в месяц среднее время ответа может быть уменьшено до 0.1 с. Наихудшая конфигурация среди рассмотренных дает время ответа около 2 с. Чтобы достичь времени ответа 0.1 с при наихудшей конфигурации может потребоваться затратить около 3400 долларов на дисплей в месяц. ■

8.4. ФАКТОРЫ ПРОИЗВОДИТЕЛЬНОСТИ В ИЕРАРХИЧЕСКОМ ПРОЕКТИРОВАНИИ

Метод итеративного проектирования, введенный в разд. 8.2 и обсуждавшийся с точки зрения его производительности в разд. 8.3, пока еще используется в большей части проектов вычислительных систем. Иерархический метод, также кратко рассмотренный в разд. 8.2, был предложен в качестве лучшей альтернативы. Основы этого метода по существу те же, что и основы аналитических подходов к проектированию первоначальной системы, описанных в разд. 8.3.1. Чистый итеративный подход к проектированию состоит в построении системы (реальной или в форме модели) на основе опыта и интуиции с последующей проверкой, работает ли она и как, и изменении этой системы до тех пор, пока она не станет приемлемой с функциональной и экономической точек зрения и с точки зрения производительности. В разд. 8.3.1 мы вместо этого попытались получить спецификации компонентов и информацию о размерах из требований на производительность и грубых моделей системы и рабочей нагрузки. Спецификации компонентов должны были затем использоваться в качестве ориентиров для выбора или проектирования компонентов оборудования и программного обеспечения первоначальной системы. Итеративные изменения первоначальной системы привели бы нас в конце концов к приемлемой системе. Заметим, что в наших простых примерах заданные модели никогда не изменялись и что первоначальная конфигурация предполагалась фиксированной. В реальной процедуре проектирования проектировщик создает несколько моделей, соответствующих различным конфигурациям, содержащим, например, много дисков или барабанов, различные типы каналов, несколько центральных процессоров, и сравнивает их характеристики «стоимость — производительность», чтобы получить полезную рекомендацию о выборе конфигурации.

Метод получения характеристик системы из проектных спецификаций может быть распространен на весь процесс проектирования, если у нас достаточно информации для такого подхода. Этот метод хорошо известен и вполне успешно применяется проектировщиками в некоторых областях техники, в которых проектируемые системы достаточно ясны. Это, по-видимому, неверно для вычислительной техники. Только несколько типов систем, в первую очередь те, рабочая нагрузка которых может быть относительно хорошо описана, могут проектироваться в соответствии с иерархическим подходом. Этот подход (или, скорее, комбинация итеративного и иерархического методов) становится все более распространенным по мере накопления знаний о вычислительных системах, пользователях машин и прикладных программах.

Чисто иерархический подход сводится, как упоминалось в разд. 8.2, к последовательному снижению уровня абстракции спецификаций системы. На каждом уровне абстракции существует спецификация системы в целом, обладающая степенью детальности, соответствующей этому уровню. Спецификация, полученная на самом нижнем уровне, должна служить основой для изготовления системы. Таким образом, иерархический подход порождает ряд моделей системы в целом, по одной на каждом уровне. Вообще говоря, можно определить функциональные характеристики и характеристики производительности каждой модели, которые должны быть совместимы с характеристиками для всей системы. На данном уровне абстракции модель будет состоять из частей, или *модулей*. Модель следующего уровня можно получить [см. Парнас (1972)] подразделением этих модулей на связанные друг с другом подмодули, функции и производительность которых должны быть специфицированы до того, как это будет сделано. Однако, как было отмечено Парнасом (1969), более низкого уровня абстракции можно достичь также, идентифицируя подмодули, общие и для других модулей, а не разделенные, как это подразумевается концепцией дробления, или заменяя модуль на более детальный, например учитывающий внешние влияния или взаимодействия с другими модулями, которые игнорировались на более высоком уровне. Когда определен новый уровень, проектировщикам следует удостовериться, что функциональные требования и требования производительности, установленные в начале этой процедуры и использованные для определения первого уровня, все еще выполняются (если только, разумеется, сами методы, использованные для перехода к новому уровню от предыдущего, не гарантируют автоматическое достижение этой цели).

Не следует путать иерархическое проектирование с понятием *системы с иерархической структурой*. Такая система организована в виде совокупности одновременно существующих *уровней*. Первый уровень (уровень 1) обычно соответствует тому, что видит пользователь. Другими словами, это интерфейс между пользователем и машиной. Все детали, скрытые от пользователей, принадлежат нижележащим уровням. Этот принцип можно распространить на все уровни. Уровень i может быть доступен непосредственно только уровню $i-1$ и сам имеет доступ только к уровню $i+1$. Модуль уровня i будет выражаться в терминах модулей уровня $i+1$ и не будет выражаться (по крайней мере явно) в терминах уровней $i+2$, $i+3$, ... Тот же самый принцип можно кратко выразить, сказав, что эти уровни *не прозрачны*. На последнем уровне мы имеем *примитивы*, являющиеся реальными компонентами аппаратуры. Многоуровневая система может рассматриваться как *иерархия виртуальных машин*. Каждый уровень является виртуальной машиной, реализованной в терминах нижележащей машины, т. е. с использованием функций, предоставляемых уровнем, лежащим непосредственно под данным. Последняя машина в этой иерархии, разумеется, должна быть физической машиной.

Иерархическую структуру можно в некоторой мере обнаружить во всех системах. Она обычно содержит по меньшей мере уровни оборудования, операционной системы, программирования и уровень языков управления заданиями. Дейкстра (1968) обнаружил, что иерархическая структура очень удобна и внутри операционной системы, особенно с точки зрения ее реализации, тестирования и изменения. Заметим,

что эти соображения согласуются с некоторыми из наиболее важных требований структурного программирования (см. разд. 9.4.3). Для получения программы с иерархической структурой рекомендуется использовать процедуры проектирования программ сверху вниз. Однако, особенно при проектировании вычислительных систем, удобно разделять эти две концепции. Можно использовать иерархическую процедуру для проектирования одноуровневой программы. Многоуровневая система в свою очередь может быть спроектирована неиерархическим методом. Мы должны, однако, понимать, что при проектировании многоуровневой системы естественнее пользоваться иерархическим подходом и что иерархические методы проектирования естественным образом приводят к многоуровневым системам, даже если соответствие между уровнями системы и уровнями абстракции не окажется однозначным.

Как и в случае многоуровневых программ, очень мало известно о производительности многоуровневых систем. В идеальном случае, спецификации производительности для уровня $i+1$ должны получаться непосредственно из спецификаций для модулей уровня i. Последнее должно также оказывать влияние на нижележащие модули. Состояние науки еще не достигло такого уровня, чтобы это было осуществимо. Однако уже существует несколько моделей, которые можно использовать для этих целей на различных этапах процедуры проектирования, и ожидается, что в результате быстрого развития области моделирования будет получено большое количество таких моделей. Следует также отметить, что разбиение системы на излишне мелкие модули может привести к неэффективности вследствие накладных расходов, возникающих за счет планирования модулей и межмодульных связей.

Как отмечал Линч (1972), иерархическое проектирование систем окажется возможным и с точки зрения производительности, только если удастся сконструировать полные модели всей системы. Эти модели первоначально должны быть настолько просты, чтобы их можно было решить. Однако они также должны допускать переход ко все более сложным и детализированным подмоделям, соответствующим все более низким уровням системы. Далее, должны существовать методы соотнесения производительности подмоделей с производительностью модели на ближайшем более высоком уровне. К сожалению, нелинейности и сложные межмодульные взаимодействия делают разложение модели на подмодели крайне затруднительным. С некоторыми перспективными направлениями дальнейшего исследования можно ознакомиться в работах Куртуа (1975) и Брауна и др. (1975). Куртуа изучал применимость метода разложения, используемого в эконометрике, к стохастическим моделям вычислительных систем. Браун и др. сконструировали иерархическую модель с очередями для сложной системы, проанализировали отдельно различные ее части, а затем объединили результаты этого анализа, чтобы получить решение для всей системы. Для проектирования многоуровневой системы было предложено также несколько правил, подсказанных соображениями производительности. В качестве примера приведем правило, упоминавшееся в работе Линча (1972): константы времени функций, реализуемых определенным уровнем, должны быть намного больше констант времени функций следующего уровня, лежащего ниже его. Другими словами, чтобы многоуровневая система была эффективна, необходимо заставить каждый модуль уровня $i+1$, если он активирован некоторым модулем уровня i, отреагировать гораздо быстрее, чем требуется от последнего модуля, когда он активирован модулем уровня $i-1$.

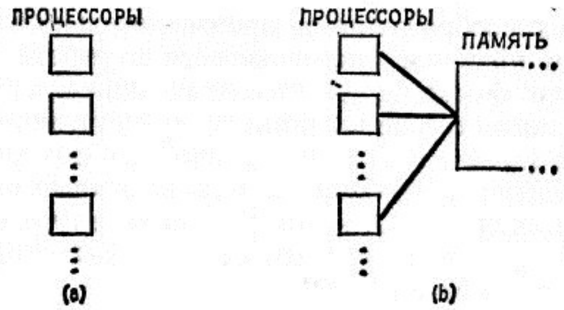
Мы уже видели, что аналитические методы не позволяют нам разработать полностью иерархическую процедуру проектирования. С другой стороны, методы моделирования, если применять их должным образом, могут оказать существенную помощь. Теперь мы, сделав упор на аспекты производительности, рассмотрим пример, в котором иерархический подход, предложенный Цюрхером и Ранделлом (1968), применен к первым уровням проектирования мультипроцессорной системы. Пример взят из работы Ранделла (1968).

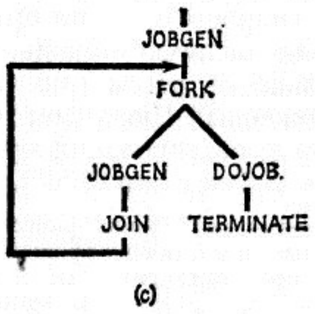
Пример 8.9. Нужно спроектировать многопроцессорную систему. Ее функциональные спецификации, а также спецификации производительности и стоимости предполагаются существующими, но не заданы явно. Методика проектирования состоит в разработке имитационной модели, которая в каждый момент времени представляет состояние проекта и, в частности, содержит принятые к этому моменту решения. Такая имитационная модель помимо того, что она является средством целенаправленного развития и документации этапа проектирования, позволяет проектировщикам проверить функциональную правильность системы и оценить ее производительность по мере ее детализации. Этот подход является иерархическим и разрабатывается от высшего уровня абстракции к низшему.

Рис. 8.12(a) является блок-схемой системы на высшем уровне абстракции. Задания, каждое из которых характеризуется временем поступления (соответственно распределению интервалов между поступлениями IAT) и временем обработки PT, появляются, обрабатываются одним из процессоров и покидают систему. Не накладывается ограничений на число процессоров, а прочие детали, например память, каналы и внешние накопители, не принимаются во внимание. Для моделирования нашей системы (которая в этот момент еще крайне проста) наиболее удобна организация моделирующей программы по принципу взаимодействия процессов (см. разд. 3.2.3). Цюрхер и Ранделл использовали расширение языка ФОРТРАН, ориентированное на моделирование процессов, так как они не располагали языком для моделирования такого типа (например, СИМУЛА). Мы в нашем описании примем этот же язык, обладающий особенностями, допускающими определение параллельных процессов. Например, на рис. 8.12(c) представлена блок-схема модели на высшем уровне абстракции. После вызова подпрограммы JOBGEN, генерирующей первое задание, FORK вызывает параллельное выполнение двух подпрограмм JOBGEN и DOJOB, или, точнее, такое выполнение, как будто бы они были вызваны и выполнены одновременно. Это отражает тот факт, что задания предполагаются независимыми, и, следовательно, они выполняются параллельно без всяких ограничений. В то время как JOBGEN, сгенерировав новое задание, возвращает с помощью операции JOIN управление в точку, где вновь активируется FORK, DOJOB разделяется всеми заданиями, которые выполняются совместно в моделируемой системе.

Функции JOBGEN и DOJOB более подробно показаны на рис. 8.12(d). Обе они вызывают программу TICTOC, моделирующую ход времени увеличением показания часов модели на величину, заданную аргументом TICTOC.

Очевидно, что на этом уровне абстракции можно исследовать только те вопросы производительности, которые относятся к числу заданий, присутствующих в системе в некоторый заданный момент времени, и ко времени, в течение которого находилось





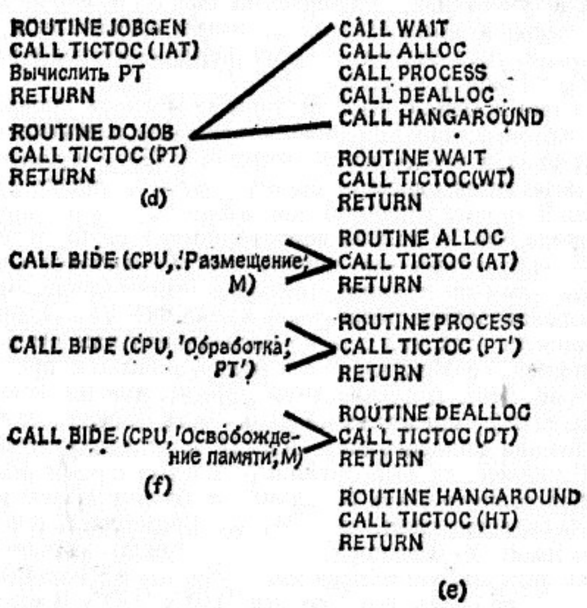


Рис. 8.12. Модель уровня 0 многопроцессорной системы: (a) — блок-схема шага 0; (b) — блок-схема шага 1; (c) — структура моделирующей программы; (d) — программы модели на шаге 0; (e) — изменения и дополнения к (d) для получения модели на шаге 1; (f) — изменения в (e), необходимые после добавления уровня 1. [*Источник*: Ранделл (1968).]

в системе некоторое задание. Могут быть вычислены времена прохождения и количество заданий, завершившихся в течение определенного интервала времени (т. е. средняя пропускная способность на этом интервале). На данном этапе значения этих

Таблица 8.6

Вероятности состояний и индексы производительности на различных уровнях иерархической модели многопроцессорной системы

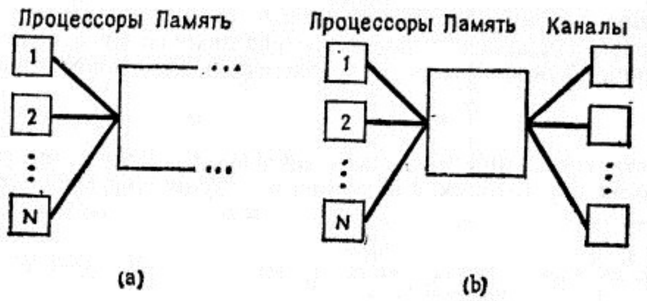
Конфигурация: 4 ЦП, 8 каналов, 512K байт
Рабочая нагрузка: среднее время ЦП — 1 с; средний размер памяти — 16K байт; экспоненциальные распределения

	При t=30 с	
	среднее	максимум
Уровень 0 [1]		
Число заданий:		
в системе	23.25	40
в состоянии ожидания	0.19	2
в состоянии размещения	16.23	31
в состоянии обработки	6.40	17
в состоянии удаления (освобождения памяти)	0.28	2
в состоянии прерванности	0.14	3
завершенных	83	83
Время прохождения задания (с)	6.37	13.79
Уровень 1 [2]		
Число заданий в пуле, которые должны быть		
размещены	0.46	5
обрабатываться	0.57	6
удалены	0.28	2
Число заданий в		
очереди на размещение	8.06	29
очереди на обработку	2.35	13
очереди на удаление	0.0	2
Число заданий в		
центральных процессорах	3.17	4
каналах	5.33	8
Использованная память (байт)	344 572.75	520 944
Коэффициент использования ЦП	0.7915	—
Уровень 2 [3]		
Число заданий в пуле каналов	2.38	18
Число заданий в прерванных ЦП	0.31	3
Коэффициент использования каналов	0.6659	—

Источник: Ранделл (1968).
[1]) См. рис. 8.12.
[2]) См. рис. 8.13.
[3]) См. рис. 8.14.

индексов, полученные на модели, будут, разумеется, оптимистическими, так как на доступность ресурсов системы не наложено никаких ограничений.

Посредством простого функционального разделения мы можем ввести дополнительные детали в эту модель (которая будет называться *моделью шага* 0). Например, мы можем расширить модель задания, заменив единственное состояние задания (сос-



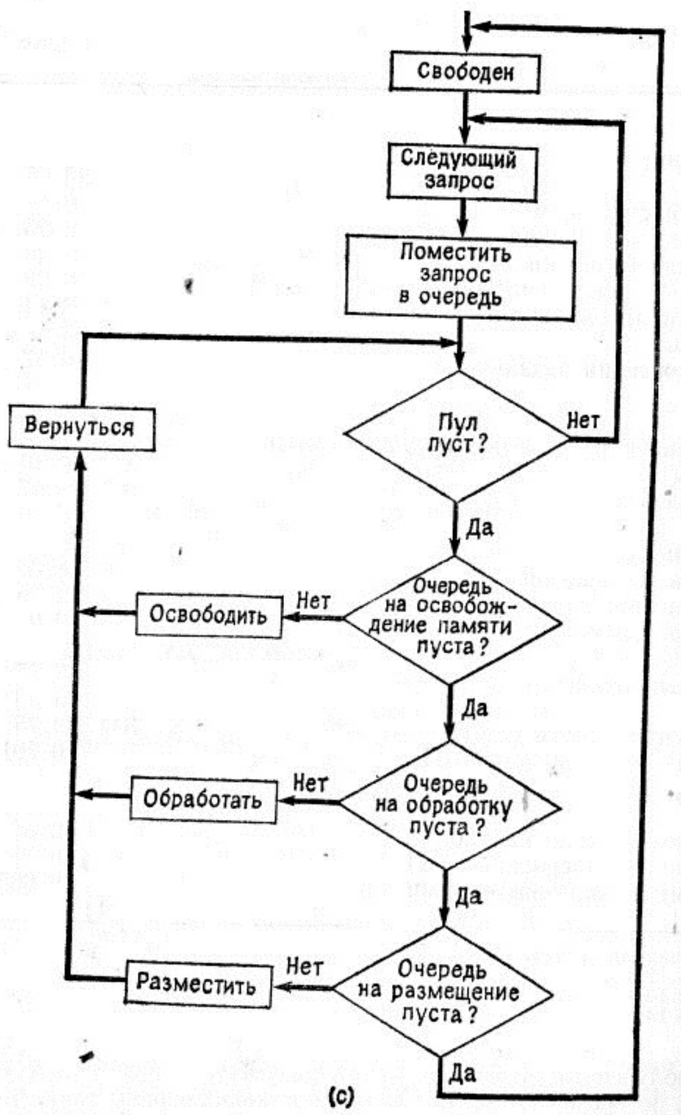


Рис. 8.13. Модель уровня 1 многопроцессорной системы: (a) — блок-схема шага 0; (b) — блок-схема шага 1; (c) — диаграмма обработки в ЦП; (d) — участок РАЗМЕСТИТЬ схемы (c) на шаге 0; (e) — участок РАЗМЕСТИТЬ схемы (c) на шаге 1; (f) — изменения в (e), необходимые при добавлении уровня 2. [*Источник:* Ранделл (1968).]

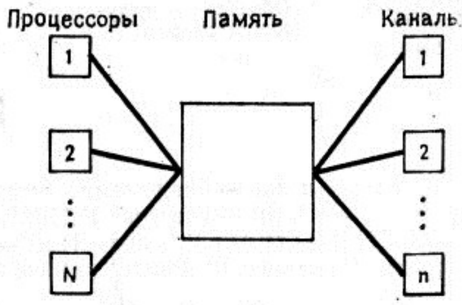
тояние обработки) на пять: ожидание, размещение, обработка, освобождение памяти, прерванность. Сделав это, мы определили память как ресурс, который может быть назначен заданию перед его выполнением и отобран перед уходом задания из системы. Однако, как показано на рис. 8.12(b), память предполагается бесконечной. Любое задание, поступающее в систему, может получить столько памяти, сколько ему нужно. Изменения модели шага 0, необходимые для перевода ее в модель шага 1, можно определить из рис. 8.12 (d) и (e). Вызов программ TICTOC в DOJOB заменяется на последовательность пяти вызовов, а пять подпрограмм, которые придется добавить, состоят просто из обращений к TICTOC с продолжительностью соответствующего состояния в качестве аргумента.

Здесь следует сделать два замечания. Во-первых, описание задания, необходимое на шаге 1, оказывается более подробным, чем на шаге 0, так как должны быть указаны продолжительности пребывания в пяти состояниях для каждого задания. Во-вторых, увеличенная степень детальности позволяет нам, используя модель, исследовать такие вопросы, как: сколько заданий размещено в памяти в определенный момент времени или в среднем? Каково максимальное число заданий, одновременно находящихся в памяти? Преобразовав модель шага 0 в модель шага 1, мы увеличили количество переменных, описывающих мгновенное состояние системы. Эти переменные перечислены вместе с упомянутыми выше индексами производительности в табл. 8.6 под заголовком «уровень 0».

Не просто ввести в полученную нами модель ограничение такого рода, как конечное число процессоров. Изменение модели не представляет непреодолимых трудностей в этом простом частном случае, но в общем случае любое изменение большой модели может оказаться дорогим и рискованным. И, разумеется, следует ожидать, что проектировщики несколько раз изменят свое мнение во время проектирования. Вместо того чтобы пытаться изменить модель, мы создадим другую модель более сложной системы. Новая модель не вытеснит старую, а будет с ней связана и будет сосуществовать с ней. Все запросы к процессорам (на размещение, обработку и освобождение памяти) будут помещаться в буфер, называемый *пулом ЦП*. Чтобы заставить старую модель делать это, мы должны только заменить обращения к TICTOC в программах ALLOC, PROCESS и DEALLOC на обращения к BIDE, как показано на рис. 8.12(f). BIDE — это программа, помещающая запросы в пул. Ее аргументами являются имя пула, тип запроса и желаемое количество запрашиваемого ресурса (например, пространство памяти *M* для размещения с последующим освобождением, время обработки PT для выполнения). Модель рис. 8.12 (d) и (e), измененная так, как показано на рис. 8.12 (e), будет называться *моделью уровня* 0.

Далее будет введен с помощью процедуры FORK набор совместно выполняющихся в ЦП процессов. Блок-схема процесса в ЦП такова, как показано на рис. 8.13 (c). Этот процесс не работает до тех пор, пока запрос не появится в пуле ЦП. Это событие

заставляет процесс ЦП добавить данный запрос к соответствующей очереди. Заметим, что введения этой очереди требует ограничение числа ЦП. Будут рассмотрены три очереди: очередь на размещение в памяти, на обработку и на освобождение памяти. Когда пул пуст, процесс ЦП проверяет эти очереди и имитирует выполнение одного из ожидающих запросов в соответствии с приоритетами, которые можно получить из рис. 8.13 (c). Процесс ЦП (вместе с соответствующим ЦП) возвращается в нерабочее состояние только в том случае, когда пул ЦП и все три очереди пусты. Обработка запроса моделируется, как показано на рис. 8.13 (d), на котором изображена блок-схема выполнения запроса на размещение в памяти. Счетчик, который фиксирует объем занятой памяти, увеличивается, вызывается программа TICTOC с аргументом, равным времени ЦП, необходимому на операцию размещения (APT), и, наконец, активируется программа REVERT.



(a)

(b)

Рис. 8.14. Модель уровня 2 многопроцессорной системы: (a) — блок-схема; (b) — диаграмма процесса в канале. [*Источник:* Ранделл (1968).]

Вновь степень детальности можно увеличить, не изменяя уровень абстракции. Мы можем добавить каналы к модели, как на рис. 8.13 (b). Первоначально число каналов будет не ограничено. Времена, необходимые каналам для загрузки задания в память (ACT) и для разгрузки его из памяти (DCT), будут вычисляться исходя из скорости канала и размера задания. Потребуется только минимальное изменение в той части процесса ЦП, которая связана с размещением и освобождением памяти. Рис. 8.13 (e) показывает такое изменение для части, связанной с размещением. Процедура FORK позволяет процессу ЦП продолжать обслуживать остальные запросы, пока активированный канал выполняет желаемый обмен.

Задание, находящееся в состоянии размещения на уровне 0 может при этом находиться в одном из четырех состояний на уровне 1: в пуле, в очереди на размещение, в процессе размещения в памяти с помощью ЦП или в процессе загрузки через канал. Аналогичные соображения с некоторыми очевидными изменениями можно высказать для состояния обработки и состояния освобождения памяти. Программы на уровне 1 работают с менее абстрактными переменными состояния, чем на уровне 0. Взаимосвязи между этими двумя уровнями таковы, что переменные уровня 0 отражают переменные уровня 1. Например, продолжительность пребывания в состоянии размещения для некоторого задания непосредственно связана с продолжительностями пребывания в четырех состояниях, на которые подразделяется это состояние на уровне 1. Переменные состояний уровня 1 перечислены в табл. 8.6 вместе с индексами производительности, которые можно вычислить на этом уровне. Введя ограничения на число процессоров и на размер памяти, можно получить значимые ответы на вопросы относительно коэффициентов использования ЦП и памяти. Также очевидно, что наличие уровня 1 влияет на индексы производительности, измеряемые на уровне 0, а именно на время прохождения и пропускную способность.

Как отмечалось Ранделлом (1968), введение другого часто встречающегося на практике ограничения, а именно ограничения на число каналов, можно произвести на уровне 1. Однако удобнее образовать еще один уровень. Можно учредить *пул каналов*, работающий аналогично пулу ЦП, описанному выше. Подпрограмма FORK будет использоваться для создания *n канальных процессов*, где *n* — число каналов [рис. 8.14 (a)]. Канальный процесс может быть построен по блок-схеме рис. 8.14 (b). Единственные изменения, которые могут потребоваться в модели на уровне 1, показаны на рис. 8.13 (f). Одно из обращений к TICTOC в блоках активации и деактивации каждого процесса ЦП [см. рис. 8.13 (*e*)] будет заменено обращением к подпрограмме DIDE, которая помещает в пул каналов запрос на пересылку. Проделав такие изменения, мы получим *модель уровня 1*. На уровне 2 можно определить еще один индекс производительности, коэффициент использования канала, и вычислить его с помощью модели. Добавлено также несколько новых состояний задания, как показано в табл. 8.6. Этот процесс последовательного функционального разложения и создания более низких уровней абстракции может продолжаться до тех пор, пока примитивы (подпрограммы самого нижнего уровня, не содержащие никаких обращений к подпрограммам, отличным от элементарных типа TICTOC) не окажутся реализуемыми физической аппаратурой или же средствами программного или микропрограммного обеспечения.

Среди преимуществ такого подхода особенно важна для проектирования гибкость полученной имитационной модели. Например, предположим, что мы превратили подлежащие обработке задания в подзадачи одного и того же задания и, следовательно, они перестали быть не зависимыми друг от друга. Природа механизмов связи между уровнями такова, что только уровень 0 должен быть изменен. Лежащие ниже уровни, если они запрограммированы правильно, корректно обработают любую комбинацию запросов. Это же верно для модели взаимодействия с терминалами. Чтобы создать ее, достаточно заменить состояние процесса на уровне 0 на последовательность состояний обработки и ожидания. Такое изменение, очевидно, потребует, чтобы в модель было введено больше информации относительно поведения задания и пользователя, однако при этом не нужно будет вносить никаких изменений в модель на более низких уровнях. ■

Иерархические подходы к проектированию систем интересны и многообещающи. Однако все еще существует несколько препятствий, затрудняющих использование этих подходов в их чистом виде. Среди перечисленных Парнасом и Сивиореком (1975) препятствий, некоторые из которых мы уже упоминали, оказались следующие:

a) Проектные требования трудно определить в общем виде и описать точно, не подразумевая решений, которые следует принимать только в ходе проектирования.

b) Множество возможных проектов, удовлетворяющих спецификациям, обычно слишком велико и должно быть ограничено добавлением внутренних спецификаций, например касающихся имеющегося оборудования.

c) Часто примитивы (например, конфигурация оборудования) известны в начале и не могут быть изменены.

d) Система, получаемая в результате, может оказаться негибкой или некоторые из ее компонентов могут оказаться трудными в изготовлении.

Таким образом, процедуры проектирования, в которых присутствуют в должной пропорции итеративные и иерархические подходы, чередующие этапы *снаружи внутрь* и *изнутри наружу*, вероятно гораздо

удобнее, чем чистые версии обоих подходов. Тогда задачей становится выбор правильной пропорции, так чтобы преимущества обоих подходов сохранились, а недостатки уменьшились или совсем исчезли.

8.5. МЕТОДЫ ОЦЕНКИ В ПРОЕКТИРОВАНИИ

В этом разделе мы подведем итоги наблюдениям, сделанным на протяжении предыдущих разделов этой главы относительно роли различных методов оценки производительности в проектировании систем.

Мы уже видели, что методы моделирования преобладают на ранних этапах проектирования над измерительными методами из-за того, что проектируемая система чаще всего не существует и, следовательно, не может быть измерена. Однако исключения из этого правила представляют практический интерес вследствие неудачной методики, посредством которой большинство систем проектировалось в прошлом, а некоторые проектируются и ныне. Этот подход, на который мы уже ссылались, состоит в том, чтобы изготовить первоначальную систему, запустить ее в работу как-нибудь, пользуясь подходящим методом проб и ошибок, а затем попытаться улучшить ее с помощью итеративной процедуры. Это, разумеется, один из двух возможных способов итеративного проектирования системы и к тому же худший. Другой подход, который мы пытались осветить в разд. 8.2 и 8.3, состоит в применении того же метода к модели системы. Последний метод лучше, так как он предоставляет проектировщикам гораздо большую гибкость, позволяет им отложить воплощение проектных решений в дорогих и жестких структурах оборудования и программного обеспечения и, следовательно, предоставляет возможности для получения лучших систем. Однако в некоторый момент работа на модели становится неудобной. Если используется аналитическая модель, этот момент наступает довольно скоро. Количество деталей, которые можно ввести в такую модель, обычно невелико и не позволяет продвинуться за пределы того, что мы назвали этапом начального проектирования. Очевидно, что имитационная модель гораздо мощнее с этой точки зрения. Однако имитационная модель вполне может оказаться столь же сложной и негибкой, как и проектируемая система. Если это так, то, вероятно, удобнее изготовить систему и переключиться на методы измерения. Этот момент, вероятно, раньше будет достигнут в проектных работах, проводимых изготовителями, чем в работах, проводимых потребителями. Таким образом, различие между двумя только что описанными итеративными методами проектирования состоит в том, в какой момент начинается изготовление прототипа. Установив, что последний метод лучше, мы выразили мнение, что изготовление следует начинать как можно позже, а методы моделирования, даже при их современном состоянии, могут предоставить проектировщикам информацию, необходимую для принятия большинства основных решений.

В контексте итеративной процедуры проектирования все типы измерений и все виды измерителей могут оказаться полезными, когда начи-

нает работать первая версия системы. Измерители, ведущие непрерывное наблюдение за системой с целью обнаружения симптомов ошибок, влияющих на производительность, должны дополняться измерителями, позволяющими объяснить некоторые непонятные явления, и измерителями, проверяющими правильность прогнозов, сделанных относительно производительности моделей. Таким образом, при проектировании могут оказаться полезными программометры, средства трассировки, аппаратные и программные измерители, протоколы и контрольные точки. Разумеется, внутренние измерители должны быть встроены в систему на ранних этапах ее создания и их согласованная работа друг с другом и с остальной частью системы требует постоянного внимания проектировщика (см. разд. 2.5). Большинство этих измерителей, если не все, будет использоваться не только на последних этапах процедуры проектирования. Одни из них будут непрерывно контролировать работу системы, другие будут зарезервированы в системе для потребностей в измерении, которые могут возникнуть, особенно для настройки системы, когда она будет установлена, и для периодической ее перенастройки (см. разд. 7.4.1).

Как уже отмечалось в гл. 3 и 4, методы измерения служат также целям калибровки и контроля моделей и получения значений параметров рабочей нагрузки. Измерение этих параметров требует, чтобы рабочая нагрузка существовала и была доступна. Поскольку это не всегда выполнимо на начальных этапах проектирования, нужно найти для измерений аналогичную рабочую нагрузку. Мы вернемся к этой проблеме в разд. 8.6.

Имитационное моделирование было описано выше как наиболее удобный метод для использования при итеративном преобразовании первоначального проекта в проект или прототип, который должен быть изготовлен. Первоначальный проект, построенный с помощью аналитической модели или на основании предыдущего опыта, будет реализован в виде имитационной модели, а модель будет итеративно изменяться до тех пор, пока она не будет удовлетворять проектным требованиям с довольно большой, но еще поддающейся контролю степенью детальности. Разумеется, вместо того чтобы сразу создавать большую и сложную модель для представления первоначальной системы, можно использовать иерархическую процедуру, подобную описанной в примере 8.9. Таким образом, первая часть проектирования будет в основном иерархической, а вторая — в основном итеративной. В другом случае эти два подхода могут быть перемешаны более сложным образом. Преимуществом иерархического подхода была бы, как отмечено в примере 8.9, бóльшая гибкость модели. Очевидно, что гибкость — самое важное свойство модели, предназначающейся для использования в проектировании.

Другим важным требованием к имитационной модели, особенно на первых поисковых этапах проектирования, является требование быть как можно более эффективной (т. е. быстрой). Это позволит сделать поиск в пространстве параметров более быстрым и дешевым. К со-

жалению, эффективность вступает в противоречие с точностью и разрешающей способностью. Это требование, по-видимому, согласуется с исходно простой и быстрой имитационной моделью, степень детальности которой постепенно увеличивается по мере принятия новых проектных решений и по мере уточнения новых аспектов конфигурации системы и ее организации. Сложная, подробная модель может быть сразу создана, если уже принят ряд решений, и поэтому, как в случае, описанном Нильсеном (1967), пространство приемлемых проектов ограничено гораздо больше, чем когда начинают с нуля. И даже в таких случаях построение большой модели в качестве первоначальной может оказаться вредным [см. Кэмпбелл и Хеффнер (1968)]. Оно может потребовать больше времени и больше материальных затрат, чем изготовление системы, и может оказаться, что систему легче понять и изменить, чем модель. Наиболее перспективен метод как для этапа усовершенствования, так и для этапа проектирования, предложенный Кимблтоном (1975). Модель, построенная по этому методу, использует известное решение приближенной аналитической модели для вычисления вклада в производительность каждого интервала времени между последовательными событиями типа «поступление задания» или «уход задания». Этот подход заметно ускоряет моделирование без видимого влияния на точность таких результатов, как время пребывания задания в системе.

Как аналитические, так и имитационные модели выполняют еще одну функцию в проектных работах. Их можно использовать «по требованию» для ответа на отдельные вопросы, возникающие в ходе работы. Если имеется глобальная модель системы, то с ее помощью на большинство этих вопросов можно получить ответы. Если это не так, устанавливается специализированная модель, которая ориентирована только на рассматриваемую частную проблему, и эта модель используется для получения ответа, который поможет проектировщикам принять решение. Таким образом, проектное исследование разбивается в некотором смысле на ряд подысследований, которые относительно легко выполняются аналитически или с помощью имитационной модели. В прошлом оправданием отказа от использования глобальных моделей служили несоответствия аналитических моделей, а также сложность и негибкость глобальных имитационных моделей общего применения. Введение иерархических процедур построения моделей обоих типов помогло изменить ситуацию. Следует постоянно иметь в виду основную проблему моделирования «по частям»: нелинейности и сложные взаимодействия между различными частями системы могут сделать этот способ моделирования или столь же сложным, как глобальное моделирование, или слишком грубым, чтобы можно было доверять результатам. По мере увеличения объема наших знаний о системе становится возможным разбить ее на модули с очень слабыми или хорошо известными взаимодействиями, так чтобы упомянутые выше подысследования могли выполняться независимо друг от друга, а их результаты легко сочетались. Ожидается [см. Линч (1972)], что это будет осуществляться

значительно легче в системах с иерархической структурой, если влияющие на производительность взаимодействия между модулями и между уровнями окажутся настолько хорошо определенными и настолько ограниченными, что будут удовлетворять требованиям к функциональным взаимодействиям, принятым в структурном программировании.

Роль методов моделирования, особенно имитационного, в иерархическом проектировании уже была проиллюстрирована в разд. 8.4. Что касается измерения, то, по прогнозу Линча, оно будет использоваться в основном для целей контроля качества. Таким образом, измерение проектных параметров и индексов производительности было бы гораздо важнее накопления детальных трассирующих записей, предназначаемых только для того, чтобы разбираться в неожиданных явлениях, если они будут обнаружены. Методы измерений все еще остаются наилучшим источником значений параметров модели и наиболее надежным методом калибровки и контроля модели.

8.6. ОПИСАНИЕ РАБОЧЕЙ НАГРУЗКИ ДЛЯ ПРОЕКТНЫХ РАБОТ

Проблема описания рабочей нагрузки, которую предстоит обрабатывать системе, является решающей в проектировании вычислительных систем. Читатель без колебаний согласится с тем, что такая спецификация абсолютно необходима. Действительно, достаточно вспомнить, что значения индексов, в терминах которых выражаются требования производительности, бессмысленны, если рабочая нагрузка, при которой они должны быть удовлетворены, не уточнена. С другой стороны, определение реальных условий рабочей нагрузки для проектируемой системы крайне затруднительно. Из трех классов проблем, рассматриваемых в этой книге (выбор, усовершенствование и проектирование) проектирование представляет собой тот класс, для которого специфицировать рабочую нагрузку, по-видимому, наиболее трудно. Это верно, если система проектируется не для специальной установки или хотя бы точно известного применения, а для обширного класса установок и применений.

В самом деле, если известна установка, проблема концептуально становится не сложнее такой же проблемы в контексте выбора (см. разд. 6.5). Нужно, чтобы были проанализированы нужды пользователей и предсказана эволюция этих нужд в ближайшем будущем. Это, несомненно, не тривиальные проблемы. Однако они гораздо проще тех, с которыми встречаются проектировщики при попытке разработать систему, которая должна удовлетворять нужды целого ряда установок. По мере увеличения размера коллектива возможных пользователей, представление о действительных вариантах рабочей нагрузки, которым будут подвергаться различные реализации системы, становится все менее четким. Поскольку качество проекта, грубо говоря, прямо пропорционально количеству доступной проектировщикам информации

о рабочей нагрузке, этот показатель будет проявлять тенденцию к ухудшению. Однако, если система проектируется для хорошо известного применения (например, управление определенным типом промышленного процесса, резервирование авиабилетов или мест в гостинице, информационный поиск), все же удается надежно оценить, возможно в параметрической форме, наиболее важные характеристики рабочей нагрузки. Таким образом, проектируются системы, которые будут работать эффективно на ряде установок после того, как они пройдут несколько естественных модификаций, в процессе приспособления к особенностям своей среды.

Еще более трудной задачей является проектирование системы *общего назначения*. Этот термин обычно указывает на то, что об ожидаемой рабочей нагрузке почти ничего не известно. Проектирование в этих условиях неизвестности кажется почти невозможным. Пессимист сказал бы, что это причина того, почему так много коммерческих систем общего назначения оказываются плохими. Оптимист, с другой стороны, нашел бы удивительным, что системы, спроектированные в столь безнадежных условиях, способны еще демонстрировать приличную производительность.

Разумеется, обе стороны и правы, и неправы. Чем меньше мы знаем о рабочей нагрузке, тем более трудным и менее успешным оказывается проектирование специфической, законченной системы. Система, специально спроектированная и настроенная для определенной установки, может быть превосходна на этой установке и немногих других, очень на нее похожих. Однако, по всей вероятности, она покажет низкую производительность в других видах окружения. Единственным удобным решением, по всей видимости, было бы установление связи между степенью специализации рабочей нагрузки и степенью специализации системы. Для рабочей нагрузки, определенной только как «нагрузка общего назначения», проектирование должно ориентироваться на создание не единственной системы, а семейства систем «общего назначения». Как мы уже видели в разд. 8.1, семейство можно рассматривать как набор компонентов оборудования и программного обеспечения, которые можно комбинировать различными способами, так чтобы некоторым образом покрыть область возможного применения и удовлетворить требования к классу установок, для которых предназначено это семейство. Таким образом, если не рассматривать этап настройки, процесс проектирования состоит из двух последовательных этапов: проектирование семейства компьютеров и его специализация. Второй этап выполняется, только если становится доступна более точная и подробная информация о рабочей нагрузке. Таким образом, часто удается достичь приемлемой производительности в соответствии с оптимистической точкой зрения. Однако пессимист, вероятно, был бы прав, утверждая, что этот удовлетворительный результат часто достигается за счет излишней затраты ресурсов из-за ограничений, введенных на этапе специализации компонентов семейства, и из-за отсутствия систематических методов проведения этого этапа.

На тех этапах проектирования, где используются методы моделирования, способ описания рабочей нагрузки диктуется моделью системы, которая в свою очередь будет определяться имеющейся информацией о рабочей нагрузке. Качество результатов, полученных с помощью модели, не может быть лучше, чем качество входных данных. Значения параметров рабочей нагрузки следует тщательно оценить или, предпочтительнее, измерить. Однако измерения при реальной рабочей нагрузке нельзя произвести до тех пор, пока система не установлена и не передана пользователям. Но это будет уже этап настройки, когда все предыдущие этапы завершены. Следовательно, значения параметров рабочей нагрузки придется определить измерениями на других рабочих нагрузках, характеристики которых предполагаются близкими к характеристикам неизвестной рабочей нагрузки, которую придется обрабатывать этой системе. Проектировщики семейства компьютеров часто используют трассирующие записи входных потоков, полученные на установках, где работают системы предыдущего семейства того же типа. Новое поколение компьютеров проектируется в предположении, что ему придется обрабатывать рабочие нагрузки предыдущего поколения. Поскольку принимаются определенные решения, какие-то изменения в записях входных потоков или в полученных на их основании параметрах должны быть следствиями этих решений. Например, если специфицируются наборы команд ЦП в новом семействе, трассирующие записи последовательностей команд должны быть преобразованы так, чтобы отражать новые наборы команд. Другим примером использования чужих рабочих нагрузок при проектировании может служить моделирование механизмов виртуальной памяти для изучения и сравнения различных стратегий управления памятью. В этом случае рабочая нагрузка обычно составляется из обращений к памяти, генерируемых программами, написанными не для проектируемой системы, а для другой машины (надеемся, с виртуальной памятью). Здесь, как и в предыдущем примере, сделано предположение, что рабочая нагрузка будет отличаться от использованной только в некоторых небольших и не очень важных аспектах. Часто такое предположение, особенно в последнем примере, оказывается не слишком далеким от реальности. В качестве довода можно привести, что большинство программ, которые будет выполнять новая система, вероятно, будет отличаться от программ, выполняемых старой системой, но различия между ними в организации обращений к памяти не повлияли бы на выбор стратегии управления памятью, даже если бы эти различия удалось учесть. Отстаивая эту точку зрения, мы среди прочего неявно пренебрегаем воздействием этой стратегии на пользователей.

Описанную нами ситуацию неопределенности в отношении рабочей нагрузки ни в коем случае нельзя назвать безнадежной. Ее можно существенно улучшить, предприняв систематическое изучение поведения пользователей. Эти исследования должны бы быть не только системно-ориентированными, но также и ориентированными на пользователя. Проектировщики должны больше знать об ожидаемой рабочей нагрузке,

если они хотят спроектировать лучшие системы. Эти дополнительные сведения можно получить, только проведя технические и психологические исследования нужд, реакций, привычек и ожиданий пользователей.

Задачи

8.1. Спроектируйте, используя итеративный подход, первоначальную конфигурацию малой системы пакетной обработки. Предположим, что конфигурация будет включать ЦП, диск с неподвижными головками, диск с подвижными головками и что рабочая нагрузка может быть охарактеризована по отношению к данной конфигурации так, как это описано в табл. 8.1. Спецификациями производительности для проектируемой системы являются: $\bar{T}_{min}=1.5$ задания/с и $\bar{t}_{max}=3$ с. В качестве первоначальной конфигурации возьмите конфигурацию, которая была получена в результате исследования, проведенного в примере 8.1, и изменяйте ее итеративно, анализируя модель после каждого изменения, чтобы определить, когда будет достигнута заданная производительность. Попытайтесь в ходе изменений системы рассмотреть также экономические аспекты, предполагая, что модель стоимости соответствует (8.4).

8.2. Придумайте процедуру, которую можно использовать для проектирования системы, подобной рассмотренной в примере 8.1, предполагая, что вместо спецификаций производительности сформулированы требования к стоимости. Пусть параметры рабочей нагрузки заданы в табл. 8.1, и пусть максимальная стоимость системы, вычисленная по (8.4), равна $C_{max}=80\,000$ долларов. Ваша цель — спроектировать систему с максимальным или почти максимальным $\bar{T}$, удовлетворяющую ограничению $C \leqslant C_{max}$. Рассмотрите как случай неограниченного $\bar{t}$, так и случай $\bar{t} \leqslant \bar{t}_{max}$.

8.3. Повторите проектирование первоначальной конфигурации из примера 8.1, используя модель открытой сети с рис. 4.16. Предположите, что средняя интенсивность поступления заданий равна $\lambda=1$ задание/с и что задания могут заканчиваться только после обслуживания их диском (т. е. $b_{21}=1$). Предположите также, что средняя скорость обслуживания Σ_4 уже определена и равна $\mu_4=3\,с^{-1}$. Единственной спецификацией производительности в этом случае является $\bar{t}_{max}=6$ с. Сравните ваши результаты (в частности, значения μ_1, μ_2, и μ_3) с полученными в примере 8.1 и обсудите причины всех заметных отличий, которые вы смогли обнаружить.

8.4. Процедура проектирования, рассмотренная в примере 8.1, определяет требуемые характеристики компонентов для заданной конфигурации. Однако построение конфигурации обычно остается среди неизвестных задачи. Подойдите к тому же проектированию, что и в примере 8.1, рассматривая какую-то иную конфигурацию. Например, вы можете рассмотреть конфигурацию с двумя независимыми идентичными барабанами или дисками, предполагая, что их вероятности доступа составляют половину от соответствующих вероятностей для единственного компонента в примере 8.1. Иными словами, вы можете предположить, что эти два барабана или диска нагружены одинаково. Как вы считаете, общая стоимость, вычисленная по формуле (8.4), увеличится или уменьшится в результате добавления еще одного устройства?

8.5. Решите задачу 8.2 для системы с виртуальной памятью, рассмотренной в примере 8.2.

8.6. Повторите процедуру проектирования, описанную в примере 8.2, предполагая, что коэффициент h_2' в формуле (8.8), относящийся к стоимости барабана, задан равенством

$$h_2'=h_{21}'\sqrt{N}$$

при $h_{21}'=2000$ долл $\cdot$ с$^{1/2}$. Такое изменение рассматривалось при обсуждении в конце примера 8.5.

8.7. Вам дано семейство кривых времени жизни, изображенное на рис. 3.8.1. Опишите, как бы вы ими воспользовались для проектирования иерархии памяти, подобной той, которая показана на рис. 8.7 и обсуждалась в примере 8.6. В качестве упражнения, которое должно многократно повторяться на протяжении такого проектирования, вычислите производительность трехуровневой системы виртуальной памяти, полученной из примеров 8.2 и 8.6, предполагая, что средняя программа, выполняемая в этой системе, ведет себя так, как показано на рис. 3.8.1.

8.8. Разработайте набор измерителей для проверки того, что иерархия памяти, определенная в примере 8.6, дает производительность, соответствующую предположениям и прогнозам ее проектировщиков. Обсудите также реальное изготовление измерителей, проблемы, с которыми вы встретитесь (помехи, память, разрешающая способность, предварительное сжатие и т. д.), и укажите, как бы вы попытались их решить.

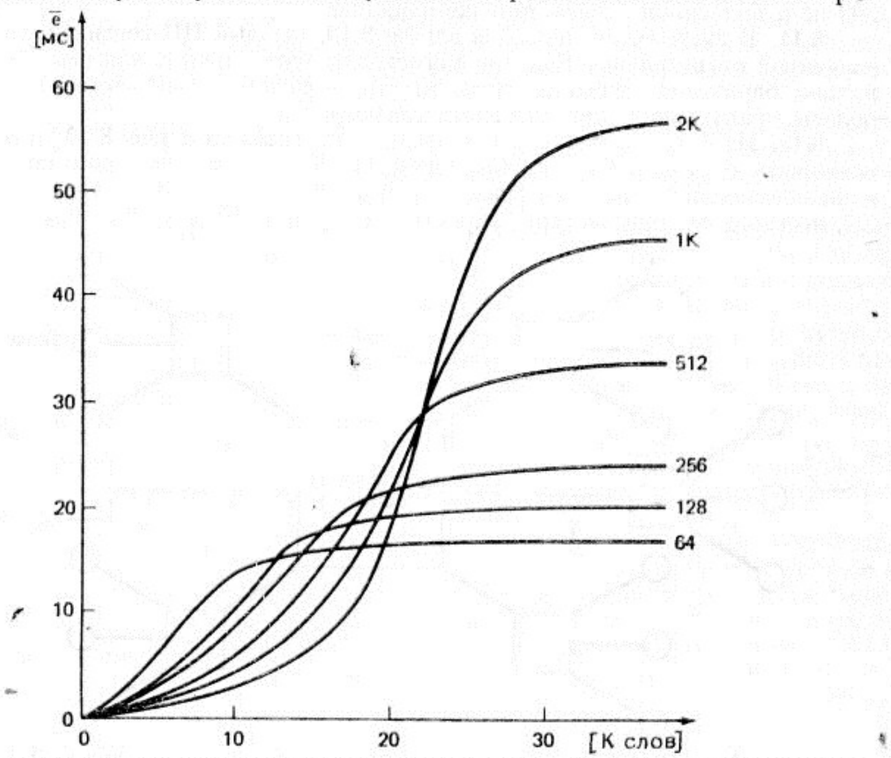


Рис. 3.8.1. Семейство кривых времени жизни для задачи 8.7. Каждая кривая соответствует своему размеру страницы (в словах), указанному около кривой.

8.9. В примере 8.6 предполагалось, что ни одна из страниц или подстраниц, которую предстоит заменить, не должна быть записана обратно на страничный барабан или в основную память. Это эквивалентно предположению, что ЦП никогда не выполняет команд, заносящих слово в кэш-память. Спроектируйте заново эту же иерархию памяти для системы, заданной в примере 8.2, но уже без такого предположения. Чтобы упростить работу, можно вероятность того, что страница или подстраница, которую предстоит удалить, должна быть записана обратно, считать постоянной и не очень чувствительной к относительно малым изменениям размера страницы или подстраницы. Пусть для подстраниц эта вероятность равна 0.6, а для страниц 0.75.

8.10. Определите приблизительную полосу пропускания памяти, необходимую для системы со следующими компонентами: два ЦП, интенсивность обращений к памяти у каждого из которых составляет 2×10^6 обращений/с; два канала, каждый из которых управляет барабаном со скоростью передачи 0.5×10^6 слов/с; два канала, каждый из которых управляет четырьмя дисками со скоростью передачи 0.3×10^6 слов/с каждый; один канал, управляющий шестью накопителями на МЛ со скоростью передачи 10^5 слов/с каждый, и один процессор обслуживания дисплея, обладающий скоростью передачи 10^6 слов/с. Повторите эти вычисления при различных предположениях о величинах вероятностей перекрытия между активными пе-

риодами различных компонентов системы. Проект «на худший случай» соответствует требованию, чтобы память была способна удовлетворить одновременные запросы всех устройств, соединенных с ней, не вызывая ощутимого снижения производительности. Как бы вы изготовили память «на худший случай» для этой системы, пользуясь полупроводниковой элементной базой, характеризующейся временем считывания около 300 нс и временами записи того же порядка?

8.11. В системе, описанной в задаче 8.10, каждый ЦП соединен с памятью независимой магистралью. Еще три магистрали используются для связи памяти с каналами барабанов, дисков и пары «ленты — дисплей» соответственно. Определите полосы пропускания для этих пяти магистралей.

8.12. Измените модель системы машинной графики с рис. 8.10(b), чтобы учесть возможные обращения к локальной памяти LS во время операций предварительной и заключительной обработки (блоки 6 и 11) и использование центрального компьютера CC, необходимое при доступе к центральной памяти CS в ходе локальной обработки

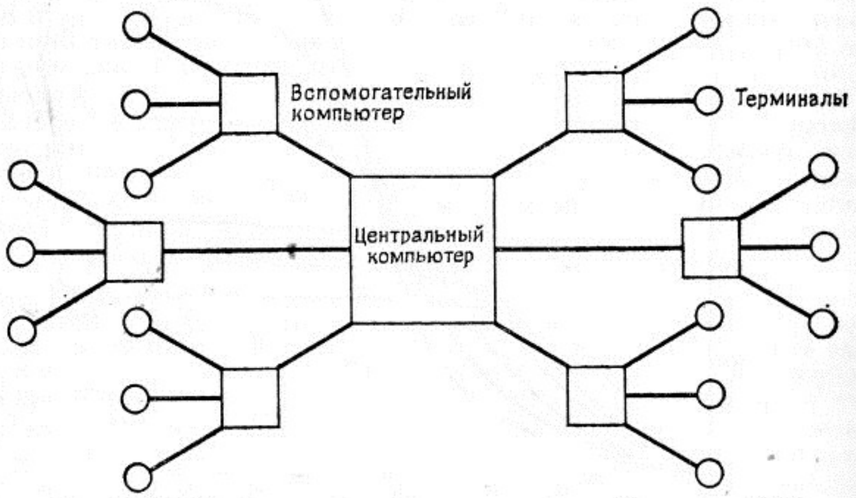


Рис. З.8.2. Блок-схема системы, использованной в задаче 8.13.

запроса на обслуживание (блок 4). Как отразились бы эти изменения на списке параметров, приведенном в табл. 8.4, и на аналитическом исследовании примера 8.8 (уравнения (8.30) — (8.43) и табл. 8.5)?

8.13. Предстоит спроектировать большую систему обработки информации, состоящую из центрального компьютера и n периферийных компьютеров, размещенных на территории целого района, для ведения банковских операций. Периферийные компьютеры соединены с центральным телефонными линиями по звездообразной схеме, как показано на рис. З.8.2. Полагают, что каждый из них обеспечивает локальное обслуживание до h диалоговых терминалов. В дневное время система должна в основном использоваться для обработки заданий, которые требуют доступа к той или иной записи оперативной базы данных, производят в ней некоторые изменения и повторно запоминают ее в базе данных. Полная база данных хранится во внешней памяти центрального компьютера, который извлекает из нее каждую ночь ряд дневных отчетов. Однако, так как обычно высока вероятность того, что задание, поступившее в определенном месте, затрагивает малую часть этой базы данных, время ответа системы зависело бы от того, хранится ли эта порция базы данных еще и в периферийном компьютере (в его внешней памяти). В этом случае, если необходимая запись не найдена в локальной базе данных, посылается сообщение центральному компьютеру. Этот компьютер знает, на каком вспомогательном компьютере находится текущая версия этой записи, просит его прислать эту запись и направляет запись к месту запроса, где запись обновляется и отсылается обратно к своему источнику. Выполняя это, центральная база данных может обновляться только раз в сутки и служит для поддержки распределенной базы данных. Пусть вероятность запроса отсутствующей записи равна $e^{-M_2/A}$, где e — основание натуральных логарифмов, M_2 — размер локальной внешней памяти в байтах, а $A = 1M$ байт. Нужно исследовать проблему выбора M_2. Используя пример 8.8 и, возможно, пример 6.1, постройте модель, которую можно было бы использовать для оценки компромисса «цена — производительность», относящегося к этому выбору, и опишите использование ее для намеченной цели. Возьмите среднее время ответа в качестве первичного интересующего индекса производительности и введите все предположения, которые сочтете необходимыми (или желательными) и разумными.

8.14. Микропрограммный ЦП может выполнять операции аппаратно, микропрограммно и программно. Следует принять решение в момент проектирования в зависимости от ожидаемых цен реализации этих трех вариантов, а также соответствующих характеристик производительности (взвешенных в соответствии с ожидаемыми частотами выполнения операций). Предполагая, что минимальный набор микрокоманд уже определен для проектируемого ЦП, опишите детально процедуру, которой вы будете следовать при решении, следует ли включить в набор микрокоманд данную не входящую в этот набор операцию, или включить ее в машинный язык путем ее микропрограммирования, или включить в библиотеку служебных программ путем ее программирования. В качестве примеров операций можно было бы рассмотреть вычисление квадратных корней, операцию поиска (найти операнд, сравнить его с содержимым регистра, увеличить счетчик или установить флажок, если удовлетворено определенное заданное условие) и деление с плавающей запятой.

8.15. Укажите, как вы оснастили бы измерителями описанную в примере 8.9 иерархическую имитационную модель для того, чтобы можно было собрать данные, перечисленные в табл. 8.6.

8.16. Модифицируйте модель, показанную на рис. 8.12 (исключая изменения, показанные на рис. 8.12(f), так, чтобы ввести ограничения на число процессоров и на размер памяти [рис. 8.13(a) и (b)] без введения нового уровня абстракции. Сравните характеристики вашего решения (легкость изготовления, понятность, гибкость) с аналогичными характеристиками модели, представленной на рис. 8.12 и 8.13.

9
Оценка производительности программ

9.1. ВВЕДЕНИЕ

Может показаться, что посвящение специальной главы оценке производительности программ противоречит принципу, который мы упоминали в нескольких местах этой книги. А именно, мы утверждали, что нет принципиальной разницы между аппаратурой и программным обеспечением и что с точки зрения оценки производительности вычислительную систему следует рассматривать как комплекс «аппаратура — программное обеспечение». С другой стороны, еще сильнее подчеркивалось отличие системы от ее рабочей нагрузки. А рабочая нагрузка системы состоит из задач по обработке информации, которые для системы выступают в роли подлежащих выполнению программ. Это отличие очень хорошо соответствует двум существенно различным и крайне важным точкам зрения: точке зрения людей, ответственных за всю систему и ее работу (администратор установки и штат его сотрудников), и точке зрения программистов (пользователей), ответственных только за качество своих программ и интересующихся лишь производительностью, с которой система выполняет их программы. Заметим, что имеется по крайней мере одна категория людей, точка зрения которых является объединением двух предыдущих: системные программисты. Систему они используют так же, как и обычные программисты, но их программы предназначены для управления ресурсами той же самой или некоторой другой системы. После отладки эти программы являются уже не частью рабочей нагрузки, а частью системы.

Таким образом, усматриваются по крайней мере две причины, причем обе достаточно серьезные, по которым в книге об оценке производительности вычислительных систем следует разобрать и вопрос об оценке производительности программ. Одна причина — это то, что рабочая нагрузка, изучение которой так важно для определения и анализа производительности системы, состоит из программ. Вторая — это то, что почти все алгоритмы, составляющие операционную систему, реализуются в форме программ, производительность которых (как программ) в общем случае существенно влияет на производительность системы.

Несмотря на тот факт, что мы часто фокусируем наше внимание на таких ориентированных на пользователя индексах производительности системы, как среднее время прохождения и время ответа, очень редко наш взгляд совпадает с ничуть не менее важным взглядом индивидуального пользователя (единственное исключение это, пожалуй, пример

из разд. 6.3.4). С его точки зрения, при оценке внимание должно концентрироваться на программе. Проектируется, выбирается и модифицируется именно программа. Система в некотором смысле проявляется лишь как фон, как немодифицируемая часть программной среды. Она, так сказать, устанавливает правила игры. На производительность программы она влияет очень сильно и иногда непредсказуемо. Выполняя программу, система оказывает пользователю различные услуги, которые ему обычно приходится оплачивать. Часто при оценке программ должны специфицироваться и явно учитываться система и ее характеристики. В других случаях, как мы увидим позднее, производительность программы может быть определена и оценена способом, не зависящим от системы.

В этой главе объектом нашего изучения будут программы. Имеется много важных характеристик, в соответствии с которыми оцениваются программы: во-первых, правильность, точность, надежность; далее, сопровождаемость программы, ее удобочитаемость, понимаемость, тестируемость, модифицируемость, обобщаемость, стоимость разработки, стоимость прохождения и т. д. Мы концентрируем наше внимание на производительности программ, т. е., грубо говоря, на том, насколько хорошо они выполняют свою задачу обработки информации. Однако следует не упускать из виду, что ценность программы зависит и от ряда характеристик, которые в некоторых ситуациях оказываются более важными, чем производительность, о чем будет рассказано в разд. 9.2. Между этими характеристиками существуют различные взаимоотношения, и при разработке и реализации программы приходится принимать различные компромиссные решения.

Мы объяснили главные причины того, что эту главу посвящаем теме оценки производительности программ. Эта тема является настолько важной частью технологии программного обеспечения и, более общо, технологии вычислительных машин, что она, безусловно, заслуживала бы целой книги. Поэтому мы ограничимся вводным обсуждением ее наиболее важных аспектов и простейшими типами программ и систем. Будут рассматриваться только последовательные нерекурсивные программы, написанные для традиционных машин. У систем с виртуальной памятью, на которые мы будем ссылаться, предполагается самая бесхитростная организация. Распределение виртуального пространства статическое, более подробно это будет объяснено позднее. Во всей главе предполагается наличие в системе некоторой техники страничной обработки и стратегии запроса страниц. Иллюстрации основных понятий, используемых в этой главе, приводятся на ФОРТРАНе, который все еще самый широко известный и распространенный язык научных и технических приложений. Выбор этих решений объясняется нашим стремлением к краткости и ясности изложения. Слишком большое количество подробностей обычно мешает усваивать идеи и понятия. Перенесение же усвоенных понятий на практический материал, связанный с более сложными или просто другими типами программ, языков и машин, является сравнительно нетрудным делом.

Структура данной главы отражает структуру второй половины книги. В разд. 9.2 мы определим, что такое программа с нашей точки зрения, и обсудим главные индексы, которые могут использоваться для количественной оценки ее производительности как в невиртуальной (разд. 9.2.1), так и в виртуальной (разд. 9.2.2) запоминающей среде. В разд. 9.3 представляются основные методы и средства, которые могут использоваться для вычисления этих индексов. В разд. 9.3.1 нас будут интересовать процесс измерения, а моделирование (как аналитическое, так и имитационное) разбирается в разд. 9.3.2. Раздел 9.4 посвящен исследованию трех главных классов проблем оценки. В соответствующих подразделах показывается, как способы оценки, разобранные в разд. 9.3, могут быть применены к проблемам выбора программ (разд. 9.4.1), проблемам усовершенствования программ (разд. 9.4.2) и проблемам проектирования программ (разд. 9.4.3).

9.2. ИНДЕКСЫ ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММЫ

9.2.1. Время и память

Что такое программа? Кнут (1968) дает следующее определение: «*Программой* называется описание вычислительного метода на языке программирования». В этом определении ключевыми терминами являются термины *вычислительный метод* и *язык программирования. Вычислительный метод* — это процедура, обладающая всеми характеристиками алгоритма, за исключением, возможно, конечности, т. е. свойства завершаемости после конечного числа шагов. Алгоритм, как известно нашим читателям, это конечный набор правил, точно определяющий последовательность операций или шагов, которые необходимо выполнить для решения некоторой задачи. Тот факт, что программа написана на языке программирования, означает, что она может быть выполнена на любой вычислительной машине, которая «понимает» этот язык. Разумеется, каждый знающий язык человек также может выполнить эту программу, например, с помощью карандаша и бумаги.

Когда программа выполняется (человеком или вычислительной машиной), она преобразует некоторую исходную информацию в выходные данные, обычно называемые *результатами*. Так, например, программа, вычисляющая квадратный корень положительного числа x, в качестве исходных данных имеет значение x и в качестве результата $\sqrt{x}$ (или приемлемое приближение $\sqrt{x}$). Эта точка зрения соответствует схеме типа «черного ящика», представленной на рис. 9.1 (a).

Группа самых интересных вопросов, касающихся программы, относится к *трудности* ее выполнения. Конечно, эта трудность зависит от различных факторов, и среди них — сложность вычислительного метода (изучением которой занимается теория сложности вычислений) и умение исполнителя. Первой реакцией на вопрос: «Как трудно было вычислить $\sqrt{x}$ с помощью карандаша и бумаги по этой программе?» мог

бы, например, быть следующий ответ: «Для $x=19.35$ и ошибки, не превышающей 10^{-3}, нам потребовалось приблизительно 3 минуты». Отсюда, по-видимому, следует сделать вывод, что указание *времени выполнения* программы является содержательным, хотя и приблизительным, способом описания ее трудности. Конечно, из приведенного диалога следует, что задающий вопрос что-то знает о наших вычислительных способностях и о вычислительных средствах (логарифмическая

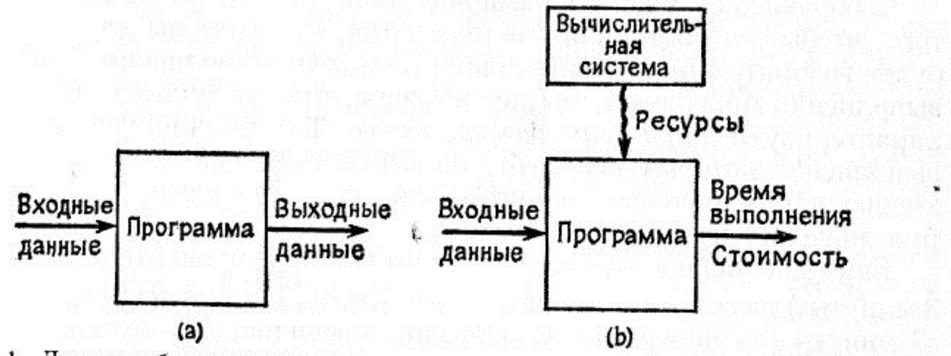


Рис. 9.1. Два способа характеристики программы как черного ящика: (a) — характеристика «вход-выход»; (b) — характеристика «вход — производительность — стоимость».

линейка, счеты, карманный вычислитель), которым мы располагаем при выполнении данной программы. Другой возможный способ охарактеризовать трудность программы состоит в указании стоимости выполнения, являющейся суммой стоимостей используемых ресурсов (наша умственная энергия, время, карандаш, бумага и т. д.). В общем случае стоимость будет зависеть и от входных данных, так же как и для случая со временем выполнения, когда мы требовали ответа «для $x=19.35$ и ошибки, не превышающей 10^{-3}».

Представление программы по схеме типа «черного ящика», соответствующее нашей точке зрения в этой главе, приведено на рис. 9.1(b). Здесь мы интересуемся не результатами программы, а в предположении, что результаты правильны, временем, которое необходимо для их получения, и тем, сколько нам нужно платить за использованные программой ресурсы. Позднее мы увидим, что оба эти индекса тесно, хотя и не всегда непосредственно, связаны друг с другом.

В то время как стоимость есть прежде всего экономический показатель, время выполнения обладает всеми признаками показателя производительности. Если его принять в качестве меры трудности, то он может использоваться для сравнения двух программ, выражающих один и тот же вычислительный метод, или двух программ, выражающих два различных вычислительных метода получения идентичных результатов. Более быстрый будет и более лучшим.

Время выполнения мы будем считать главным индексом производительности программ, но никогда не следует забывать, что в иных ситуациях и средах более важными могут оказаться другие индексы. Своей

популярностью индекс «время выполнения» обязан также тому, что он тесно связан со стоимостью выполнения программ. Впрочем, иногда эти два индекса противоречат друг другу. В таких случаях выбор индекса делается в соответствии с критерием минимума глобальной стоимости. Следует стремиться к оптимальному компромиссу между временем, затраченным программистом, и временем, затрачиваемым вычислительной машиной, учитывая, разумеется, их относительные цены.

Соображение, уже высказанное нами ранее, но заслуживающее того, чтобы его повторили, состоит в том, что хотя мы для краткости будем говорить о производительности, времени выполнения и стоимости выполнения программы, мы не забываем, что эти индексы фактически характеризуют *пару (программа, вход)*. Только для тех программ, выполнение которых относительно нечувствительно к изменению исходных данных, можно говорить о производительности, не специфицируя значения исходных данных.

Еще одно важное замечание состоит в том, что наш ответ («примерно 3 минуты») на вопрос о трудности вычисления $\sqrt{x}$ неявно подразумевает «3 минуты без перерыва». Совершенно очевидно, что нельзя характеризовать трудность выполнения программы длительностью периода, в течение которого производилась деятельность, не имеющая прямого отношения к выполнению самой программы. В идеальном случае индекс производительности программы не должен зависеть ни от чего другого, кроме поведения самой программы. Еще можно допустить зависимость от значений некоторых системных параметров, таких, как скорость ЦП, но совершенно недопустима зависимость от таких изменчивых и непредсказуемых величин, как параметры рабочей нагрузки системы. Поэтому, как отмечалось в разд. 6.3.4, время выполнения программы в мультипрограммной среде не может считаться надежным индексом. Когда программа с заданными исходными данными обрабатывается в однопрограммном режиме, изменчивость времени ее выполнения не равна нулю [см., например, Белл (1974)], но часто достаточно мала, и время выполнения может служить приемлемым индексом производительности. Заметим также, что время выполнения программы в однопрограммном режиме в некоторой системе является нижней границей времен выполения этой программы в этой же системе в мультипрограммном режиме.

В примере 9.1 мы увидим, что имеются различные определения времени однопрограммного выполнения, отражающие различную степень зависимости от системы. Впрочем, в любом случае мы должны помнить, что, вообще говоря, время однопрограммного выполнения — это системно-зависимый индекс производительности. В принципе нет никаких гарантий, что результат сравнения производительности двух эквивалентных программ останется корректным и для других систем, отличных от той, на которой производилось сравнение. Другими словами, совершенно очевидно, что из двух программ, вычисляющих $\sqrt{x}$, более быстрая на системе A может оказаться более медленной на системе B.

Конечно, мы предполагаем, что та и другая программы могут выполняться на обеих машинах. Проблема системно-независимой характеризации программ, представляющая большое значение для оценки систем и, в частности, для выбора систем, обсуждалась в разд. 5.2.3 и 6.5 и будет снова неоднократно возникать в этой главе.

Хотя мы и идентифицировали приемлемый индекс производительности для программ (по крайней мере для программ, написанных для выполнения на данной системе), мы все еще далеки от нашей цели. Мы хотим научиться манипулировать значением этого индекса так, чтобы при разработке программы удовлетворять спецификациям производительности и улучшать качество выполнения данной программы. Поэтому от схемы типа черного ящика, которую мы использовали для определения индексов производительности и экономичности, нужно отказаться. Нам теперь необходим более детальный анализ структуры и поведения программ.

Главный вопрос при анализе производительности программы: как программа расходует время своего выполнения? Чтобы ответить на этот вопрос, полезно определить множество отдельных состояний, в которых может находиться программа во время своего выполнения. Если $S=(S_1, S_2, \ldots, S_n)$ — такое множество, то поведение программы может быть описано последовательностью состояний, в которых находилась программа при выполнении, и соответственно временем нахождения в каждом из этих состояний. Формально это описание можно представить как последовательность $\{i_j(t_j)\}|j=1, 2, \ldots; i_j \in (1, 2, \ldots, n)]$, где t_j — продолжительность j-го интервала выполнения в состоянии S_{i_j}. Если для всех j и $k=1, 2, \ldots, n$ мы определим

$$\begin{aligned} t_j(k) &= t_j, \text{ когда } k = i_j, \\ t_j(k) &= 0, \text{ когда } k \neq i_j, \end{aligned} \tag{9.1}$$

то общее время, потраченное в состоянии S_i во время выполнения программы, равняется

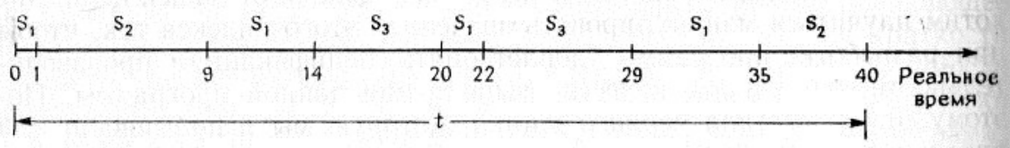
$$t(i) = \sum_{j=1}^{r} t_j(i), \tag{9.2}$$

где r — число интервалов, на которое разделяется все время выполнения (предполагается, что оно конечно). Общее время выполнения определяется по формуле

$$t = \sum_{i=1}^{n} t(i). \tag{9.3}$$

Примером такой декомпозиции является случай, когда состояния программы соответствуют различным ресурсам (машинного оборудования), взаимно исключающими услугами которых пользуется работающая программа. Если мы предположим, что система состоит из ЦП, лентопротяжного устройства и терминала и не позволяет программам производить вычисления с перекрытиями обращений к вспомогательной

памяти и операций ввода-вывода, то можно определить три состояния: состояние ЦП, состояние «лента» и состояние «терминал». Поэтому поведение программы может быть представлено так, как на рис. 9.2(a) или, эквивалентно этому, как на рис. 9.2 (b). Заметим, что в состоянии «лента» или «терминал» ЦП может активно управлять передачей данных, но мы предполагаем, что для пользователя он непосредственно никакой полезной работы не делает. Декомпозиция нашего основного



(a)

S_1: ЦП S_2: Терминал S_3: Лента

$\{i_j(t_j)\} = 1(1), 2(8), 1(5), 3(6), 1(2), 3(7), 1(6), 2(5).$

$t(1) = 1+5+2+6 = 14$
$t(2) = 8+5 = 13$
$t(3) = 6+7 = 13$
$\Big\}\ t = t(1) + t(2) + t(3) = 40$

(b)

Рис. 9.2. Декомпозиция времени выполнения программы в однопрограммном режиме с заданными исходными данными: (a) — временная диаграмма выполнения; (b) — состояния программы, их последовательность и их продолжительность.

индекса производительности t на отдельные компоненты, которые можно считать частными индексами производительности, приводит к описанию работы программы в терминах *запроса ресурсов* (например, последовательности интервалов ЦП, ленты или терминала или *периоды активности*) и *глобальных запросов ресурсов*, таких, как $t(1)$, $t(2)$ и $t(3)$ на рис. 9.2 (b). Такое описание близко описанию, на котором обычно основываются расчеты стоимости выполнения программы. В большинстве случаев использование ресурсов, которые в каждый конкретный момент могут потребляться только одной программой, оценивается в соответствии с общей продолжительностью использования. Если некоторый ресурс используется программой вместе с другим ресурсом, мы можем учесть это при определении состояний программы. Например, если ЦП и лента работают одновременно при выполнении операций переписи «лента — память» посредством механизма прямого доступа к памяти (DMA), то можно определить следующие четыре состояния: только ЦП, только лента, ЦП — лента и терминал.

Ресурсы памяти, и в частности оперативной памяти, оцениваются не только в соответствии со временем, в течение которого они используются, но также и в соответствии с занимаемым объемом. Таким образом, для характеристики требований программы к памяти машины используются средняя или максимальная *память и произведение память — время* (определяемое как интеграл от размера общей оперативной памяти по времени однопрограммного режима). В системах с невиртуальной памятью зависимость однопрограммного времени выполнения от требуемой памяти машины относительно слабая. Она была бы равна нулю, если бы не учитывались времена загрузки и разгрузки и если бы перекрытия были исключены из рассмотрения. Впрочем, накладываемые на память машины ограничения всегда влияют на общую сумму оплаты и на время мультипрограммного выполнения (когда имеется тенденция откладывать загрузку больших заданий). Совершенно по другому выглядит эта ситуация в системах с виртуальной памятью, что будет показано в разд. 9.2.2.

Очевидно, что возможны и другие типы декомпозиции, приводящие к другим индексам производительности. Например, составляющая программу информация может быть разделена на части, которые мы назовем *блоками*. В любой заданный момент выполнения программы обрабатывается один из этих блоков. В течение любого интервала времени обычно обрабатывается только некоторое подмножество блоков. В этом случае состоянием программы можно считать ссылку на некоторый блок или на некоторое конкретное подмножество блоков; время выполнения можно разбить на интервалы, соответствующие каждому из этих состояний. Блок может состоять из одного слова или символа (команда или элемент данных), группы слов, подпрограммы, массива, записи, файла, модуля операционной системы, вызываемого программой, перекрытия и т. д. Может оказаться полезным сгруппировать в один и тот же блок всю информацию, которой обмениваются за один прием оперативная память и любое другое устройство; поэтому времена доступа к устройству и передачи данных можно рассматривать как потраченные программой на доступ к данному блоку. Последовательность состояний, определяемая блоками, является описанием *схемы обращений*, производимой программой при обработке заданных исходных данных. К такому описанию мы вернемся в разд. 9.2.2.

При декомпозиции можно ограничиться отдельными частями общего времени выполнения или производить ее с большей степенью детализации. Например, нас может заинтересовать анализ того, как программа тратит время ЦП. Иначе говоря, какие виды команд выполняет программа наиболее часто? Какая часть общего времени ЦП уходит на выполнение различных видов команд или операторов? Обработка каких переменных, массивов и структур данных потребляет бо́льшую часть времени ЦП? Ответы на все эти вопросы можно получить, если с каждой командой (или оператором) в программе связать отдельное состояние и интервалы времени ЦП, затрачиваемые на выполнение программы, разбить на интервалы, затрачиваемые в отдельности в каждом из этих состояний. Последовательность $\{i_j(t_j)\}$, которая получается таким образом, мы будем называть *трассировочной записью для программы*. Практически на все вопросы, касающиеся поведения программы во время ее выполнения, можно получить ответ, если известны трассировочная запись и листинг программы. В следующем примере

это иллюстрируется применительно к некоторым полезным характеристикам программы.

Пример 9.1. В табл. 9.1 приведены исполнимые операторы небольшой программы на ФОРТРАНе. Чтобы упростить рассуждения, предположим, что программа выполняется на ФОРТРАН-машине (например, на машине, представляющей собой вычислительную систему с интерпретатором языка ФОРТРАН).

Таблица 9.1

Листинг и профиль небольшой программы на ФОРТРАНе

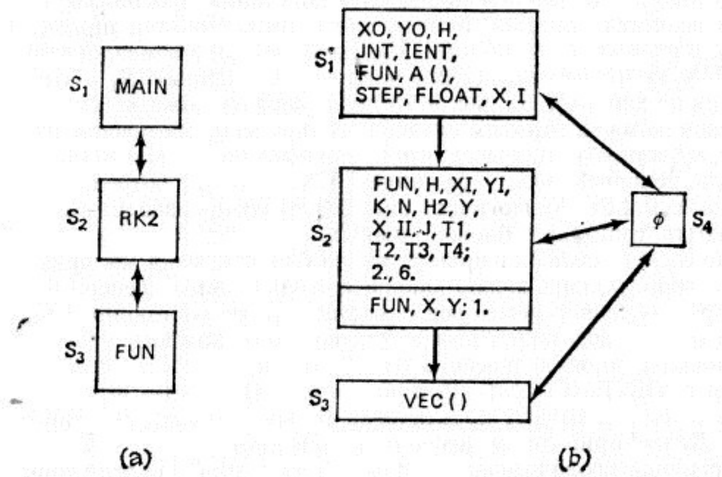
Номер оператора		Оператор	Счетчик частоты	Время
1	10	READ(5, 1) XO,YO,H,JNT,IENT	2	100
2		IF(IENT) 20,40,20	2	4
3	20	WRITE(6,2) H,XO,YO	1	50
4		CALL RK2(FUN,H,XO,YO,JNT,IENT,A)	1	2
5		STEP = FLOAT(JNT)*H	1	11
6		X = XO	1	1
7		DO 30 I = 1,IENT	1	2
8		X = X + STEP	1	2
9	30	WRITE(6,3) X,A(I)	1	51
10		GO TO 10	1	1
11	40	STOP	1	1
		END	—	—
12		FUNCTION FUN(X,Y)	8	120
13		FUN = 1./X	8	64
14		RETURN	8	40
		END	—	—
15		SUBROUTINE RK2(FUN,H,XI,YI,K,N,VEC)	1	15
16		H2 = H/2.	1	8
17		Y = YI	1	1
18		X = XI	1	1
19		DO 2 II = 1,N	1	2
20		DO 1 J = 1,K	1	2
21		T1 = H*FUN(X,Y)	2	18
22		T2 = H*FUN(X + H2,Y + T1/2.)	2	36
23		T3 = H*FUN(X + H2,Y + T2/2.)	2	36
24		T4 = H*FUN(X + H,Y + T3)	2	22
25		Y = Y + (T1 + 2.*T2 + 2.*T3 + T4)/6.	2	46
26	1	X = X + H	2	6
27	2	VEC(II) = Y	1	3
28		RETURN	1	5
		END	—	—

Источник: Кнут (1971), с разрешения John Wiley & Sons Ltd.

Можно получить описание типа черного ящика, такого, как на рис. 9.1 (а), если перечислить все входные переменные (XO, YO, H, JNT, IENT) и все выходные переменные [H, XO, YO, X, A(I)], содержащиеся в тексте программы. Для определения основного индекса производительности нам нужно специфицировать критерий, в соответствии с которым определяются начало и конец выполнения.

Имеются по крайней мере два разумных определения времени выполнения, соответствующие нашему контексту. Первое отражает точку зрения извне: время выполнения — это время между моментом, когда читающее устройство выбирает первую карту программы, и моментом, когда на выводном устройстве будет отпечатана последняя строка результатов. Это определение, более или менее соответствующее определению общего времени прохождения в однопрограммной пакетной системе, включает в период выполнения длительности различных операций, которые часто зависят больше от системы, чем от программы. Такими операциями являются, например, перепись с читающего устройства на ленту и с ленты на печатающее устройство, перекодировка, операции загрузки и разгрузки. Второе определение учитывает лишь «чистое» время выполнения. Предполагается, что программа загружена в память, и время ее выполнения есть интервал между началом выполнения первой команды и завершением выполнения команды STOP. Хотя это определение слабее отражает время ожидания программиста, у него есть то преимущество, что оно меньше зависит от системы и поэтому сильнее подчеркивает существо поведения программы. В данном примере и во всей этой главе мы будем пользоваться вторым определением времени выполнения. Заметим, что имеются и другие способы определить время выполнения программы даже в среде пакетной обработки, какой мы пользуемся здесь. Конечно, при выборе определения нужно согласовываться с типом системы (см. например, задачу 9.1).

Вернемся к нашей программе. Только что определенное «чистое» время выполнения будет суммой времен, требуемых для обработки каждой из ν вводных карт, находящихся перед картой с IENT=0. Поэтому мы могли бы интерпретировать работу программы как последовательное объединение ν прогонов, каждый из которых характеризуется своими значениями пяти перечисленных выше входных переменных. Каждый прогон начинается и заканчивается выполнением оператора READ. Прогон с номером $(\nu+1)$ прекратится сразу же после оператора READ и последующей проверки IF. Разделив суммарное время выполнения на ν, мы получили бы среднее время выполнения каждого просчета. Если набор исходных данных считается достаточно большим и может рассматриваться как «типичный», обратная величина среднего времени может быть определена как *средняя интенсивность обработки* множества исходных данных; этот индекс производительности программы часто используется для некото-



$\{i_j(t_j)\}$ = 1(104), 2(31), 3(28), 2(18), 3(28), 2(18), 3(28), 2(11), 3(28), 2(41), 1(121).

(c)

$\{i_j(t_j)\}$ = 1(104), 2(54), 4(5), 2(41), 4(5), 2(41), 4(5), 2(34), 4(5), 2(33), 3(3), 4(5), 1(67), 4(1), 1(52), 4(1).

(d)

Рис. 9.3. Состояния программы и их продолжительность для программы из табл. 9.1: (а) — состояния блоков команд; (b) — состояния блоков данных; (c) — последовательность состояний блоков команд во время экспериментального выполнения; (d) — последовательность состояний блоков данных во время того же самого выполнения.

рого типа программ (он измеряется, например, в количестве платежных ведомостей, обработанных за минуту, накладных, подготовленных за минуту, почтовых адресов, отпечатанных за час, генераций отчетов за час и т. д.).

Так как мы постулируем существование машины ФОРТРАН, которая выполняет программу и физическая реализация которой не специфицируется, нельзя определить состояния программы, соответствующие физическим ресурсам. (Фактически, когда объектный код, получаемый в результате компиляции программы, обрабатывается на оснащенной лентами системе пакетной обработки, время выполнения складывается из чередующихся интервалов ЦП и ленты. Интервалы ленты соответствуют выполнению операторов READ и WRITE.) Впрочем, состояния, соответствующие *ресурсам машины ФОРТРАН*, можно определить так, как мы сделали в примере 6.3 из разд. 6.3.4. Таблицы 6.10 и 6.11 иллюстрируют анализ некоторой программы на языке ФОРТРАН, проведенный в соответствии с ресурсами языка ФОРТРАН, перечисленными Морганом и Кэмпбеллом (1973) (см. табл. 6.9). Такой же анализ для программы из табл. 9.1 предлагается читателю в качестве упражнения в задаче 9.3.

С другой стороны, мы можем определить состояния программы как блоки, на которые подразделяется адресная память программы. Простой пример такого подразделения приведен на рис. 9.3 (a) и (b). Имеется три блока команд (главная программа, функция FUN, подпрограмма RK2) и четыре блока данных (S_4 — состояние, в котором нет ссылок на данные). На рис. 9.3 (c) показаны возможные последовательности состояний блоков команд. Анализ суммарного времени, затрачиваемого в каждом блоке, наводит, в частности, на мысль, что время выполнения может быть значительно сокращено, если ускорить работу функции FUN, которая потребляет 34.4% всего времени (см. разд. 9.4.2). Аналогично, рис. 9.3 (d) воспроизводит соответствующую последовательность состояний блоков данных.

При дальнейшем разделении программы на блоки команд мы приходим к большей характеризации трассировочной записи для программы. Теперь каждый исполнимый оператор определяет некоторое самостоятельное состояние. Трассировочная запись указывает порядок появления и длительность каждого из этих состояний. В табл. 9.2 приводится пример трассировочной записи, полученный для случая, когда на первой карте IENT=1, JNT=2, а на второй IENT=0. Зная трассировочную запись, можно получить ответы практически на все вопросы, связанные с производительностью. Далее приводится несколько примеров.

Сколько операций ввода-вывода совершает программа? На этот вопрос, который в терминах машинного оборудования соответствует вопросу о числе обращений к ленте, можно ответить, подсчитав, сколько раз выполняются операторы READ и WRITE. В трассировочной записи в табл. 9.2 показано, что это число равно 4. Аналогично можно определить, например, сколько раз вызывается подпрограмма RK2 (1 в табл. 9.2); сколько раз производится обращение к переменной XO для чтения и записи (5 в табл. 9.2); сколько времени тратится на выполнение функции FUN (224 единицы времени в табл. 9.2). На все эти вопросы легко ответить, если известны *счетчики частоты* каждого оператора, в которых подсчитывается суммарное число выполнений оператора. Набор счетчиков частоты всех операторов программы Кнут (1971) назвал *профилем программы*. Профиль программы изменяется, когда изменяются значения исходных данных. Поэтому, как и все индексы производительности, которые мы вводили, он скорее характеризует пару (программа, входные данные), а не саму программу. В табл. 9.1 приводится профиль, полученный по трассировочной записи из табл. 9.2. Заметим, что такие операторы, как оператор 21, содержащие обращение к функции, появляются в трассировочной записи для программы дважды для каждого выполнения.

Профиль программы можно также использовать для получения *динамической смеси операторов программы* (см. разд. 5.3.4). Производится классификация операторов, согласно которой каждый оператор принадлежит одному (или иногда более чем одному) классу. Сумма значений счетчиков частот операторов в каждом классе дает относительную частоту этого класса. В последнем столбце табл. 6.10 приводится смесь операторов программы из этой таблицы.

Не на все вопросы можно ответить, зная только профиль программы, который характеризует выполнение программы в сжатой форме, но с существенной потерей информации. Например, какова длина интервалов между последовательными операциями ввода-вывода? Для ответа на этот вопрос требуется знание последовательности выполнения операторов, которая содержится в трассировочной записи, но не содержится в профиле и смеси. Для трассировочной записи из табл. 9.2 ответ будет 2 единицы

Таблица 9.2

Трассировочная запись для программы из табл. 9.1

MAIN	RK2	FUN	MAIN	RK2	FUN
1(50)				21(2)	
2(2)					12(15)
3(50)					13(8)
4(2)					14(5)
	15(15)			21(7)	
	16(8)			22(11)	
	17(1)				12(15)
	18(1)				13(8)
	19(2)				14(5)
	20(2)			22(7)	
	21(2)			23(11)	
		12(15)			12(15)
		13(8)			13(8)
	21(7)	14(5)			14(5)
	22(11)			23(7)	
				24(4)	
		12(15)			12(15)
		13(8)			13(8)
	22(7)	14(5)			14(5)
	23(11)			24(7)	
				25(23)	
				26(3)	
		12(15)		27(3)	
		13(8)		28(5)	
	23(7)	14(5)	5(11)		
	24(4)		6(1)		
			7(2)		
		12(15)	8(2)		
		13(8)	9(51)		
	24(7)	14(5)	10(1)		
	25(23)		1(50)		
	26(3)		2(2)		
			11(1)		

времени между операторами 1 (READ) и 3 (WRITE), 443 единицы времени между 3 и 9 (WRITE) и 1 единица времени между 9 и 1.

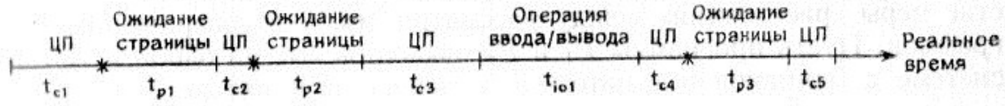
И наконец, отметим, что профиль или смесь могут быть использованы для подсчета общего времени выполнения только в том случае, когда известны времена выполнения каждого оператора или класса операторов. Конечно, оператор или класс операторов, требующие больше времени для выполнения, могут сильнее влиять на производительность программы, если даже значения их счетчиков частоты меньше. Например, оператор 3 из табл. 9.1 влияет сильнее, чем оператор 14, хотя значения их счетчиков частоты соответственно 1 и 8. Эту информацию можно получить из последнего столбца табл. 9.1, где фиксируется общее время, затрачиваемое на выполнение каждого оператора. Сумма этих отрезков времени равняется общему времени выполнения.

9.2.2. Индексы поведения программы

В среде с виртуальной памятью производительность программы и стоимость ее прохождения зависят от таких аспектов поведения программы, которые в системах с невиртуальной памятью практически не имеют значения. Здесь время выполнения гораздо более чувствительно к объему необходимой для программы памяти, чем в ситуациях, подобных рассмотренной в разд. 9.2.1. Впрочем, дело заключается не в общем объеме требуемой памяти, а в том минимальном количестве физической памяти, которое обеспечивает «эффективную» работу программы. Чтобы пояснить то, что мы имеем в виду, проведем простой мысленный эксперимент. Во время выполнения заданной программы в мультипрограммной системе с виртуальной памятью будем фиксировать количество памяти, которое выделяется программе, в каждый момент очередного начала работы ЦП. Затем мы выполним эту же самую программу на той же самой системе, но уже в однопрограммном режиме, выделяя в каждый момент ее времени ЦП такое же количество оперативной памяти, какое она имела и в мультипрограммном режиме. Время выполнения, которое мы замеряем в этих условиях, есть наш индекс производительности для программ. Соотношения компонент этого времени и стоимость прохождения практически те же самые, как и в разд. 9.2.1. Заметим, что если решение вопросов выделения памяти зависит от общей нагрузки, то на значение нашего индекса эта нагрузка будет влиять и в контексте проводимого однопрограммного эксперимента. Предложенная Бардом (1973) характеризация влияния нагрузки с помощью одного параметра позволяет приблизительно предсказывать поведение программы в условиях глобальной стратегии некоторого класса. Впрочем, в дальнейшем мы будем предполагать, что выделение программе памяти происходит независимо от любых других активных программ. Например, объем памяти может быть фиксирован и постоянен (стратегия фиксированного выделения) или в каждый момент совпадать с объемом рабочего множества программы в этот момент (см. раздел 7.4.2). Мы еще предположим, также для упрощения рассуждений, что виртуальная память реализована методом страничной обработки, кратко описанной в примерах 2.7 и 7.3.

При описании выполнения программы в терминах интервалов использования ресурсов, как на рис. 9.2, появляется новое состояние, не существующее в системах с невиртуальной памятью: *состояние ожидания страницы* (рис. 9.4). Общее время, затраченное в состоянии ожидания страницы, равняется произведению среднего времени ожидания страницы на общее число страничных отказов, происшедших за время выполнения программы.

Среднее время ожидания страницы — это системный параметр; он равен сумме двух характеристик комплекса «страничный канал — устройство»: среднего времени доступа и времени передачи страницы. Но число страничных отказов зависит как от системы, так и от программы. Система влияет на него через стратегию управления памятью, которая определяет, какой объем памяти должен быть выделен программе и какая часть программы будет резидентной в каждый момент ее выполнения. Программа на число страничных отказов влияет тем, как она организует ссылки на информацию, хранимую



*: Страничный отказ

(a)

n_f: Число страничных отказов

n_{io}: число операций ввода-вывода

$t_c = \sum_{j=1}^{n_f+n_{io}+1} t_{cj}$: время ЦП (или виртуальное)

$t_p = \sum_{j=1}^{n_f} t_{pj}$: время ожидания страницы

$t_{io} = \sum_{j=1}^{n_{io}} t_{ioj}$: время ввода-вывода

$t = t_c + t_p + t_{io}$: время выполнения

$t_{nv} = t_c + t_{io}$: время выполнения для невиртуальной памяти

Средние интенсивности страничных отказов:

$$\bar{f}_c = \frac{n_f}{t_c} \qquad \bar{f}_{nv} = \frac{n_f}{t_{nv}} \qquad \bar{f} = \frac{n_f}{t}$$

(b)

Рис. 9.4. Выполнение программы в однопрограммной системе с виртуальной памятью: (a) — временная диаграмма; (b) — возможные определения интенсивности страничных отказов.

в области виртуальных адресов. Другими словами, число страничных отказов зависит также и от *ссылочного поведения* программы.

Как было отмечено в разд. 8.3.2, задачей алгоритма автоматического управления иерархией памяти является подготовка в оперативной памяти информации, требуемой ЦП, в точности к моменту, когда ЦП ссылается на эту информацию. Если бы эта задача могла быть решена полностью, то выполнение программы должно было бы приостанавливаться только для операций ввода-вывода, явно запрашиваемых программой, так же как и в невиртуальной среде. Для достижения этой цели алгоритм управления должен пытаться предсказывать ссылочное поведение работающей программы. Рассматривая эту проблему с противоположной точки зрения, можно сказать, что программа с заданным алгоритмом управления должна попытаться приспособить свое поведение к поведению, которое ожидается от нее этим алгоритмом, так чтобы прогнозирование алгоритма стало как можно более удовлетворительным. Из этого замечания следует, что числовым показателем ссылочного поведения программы может служить индекс, показывающий, насколько близко поведение программы

к поведению, постулируемому данным алгоритмом управления. Если бы прогнозирование было совершенным, то все страницы, которые понадобятся в ближайшее время, могли быть перезагружены, и поэтому страничные отказы были бы исключены. Таким образом, *число страничных отказов* (или *средняя интенсивность отказов*) для программы, управляемой данным алгоритмом, может быть взято в качестве меры «расстояния» между рассматриваемой и совершенной программой. Пусть программы P_1 и P_2, выполняемые на одной и той же системе с одинаковой стратегией управления памятью, генерируют две средние интенсивности отказов $\bar{f}_1$ и $\bar{f}_2$, причем $\bar{f}_1 > \bar{f}_2$. В этом случае мы могли бы сказать, что для данной стратегии у программы P_2 лучшее (ссылочное) поведение, чем у P_1. Заметим, что мы предполагаем, что P_1 и P_2 работают в условиях мысленного эксперимента, описанного в начале данного раздела. Средние интенсивности отказов можно определить не только по отношению к времени выполнения в невиртуальной среде, а также и по отношению к времени ЦП или общему времени выполнения каждой программы. Эти различные определения среднего отказа иллюстрируются на рис. 9.4; среди них $\bar{f}_c$ является, по-видимому, наименее системно-зависимым индексом.

Итак, даже при соблюдении большой осторожности, сильная зависимость от стратегии не всегда позволяет успешно использовать интенсивность отказов в качестве индекса производительности программы, так как не очень ясно, что в нашем контексте означает выражение «та же самая стратегия». Например, предположим, что выбрана стратегия фиксированного распределения; означает ли это, что P_1 и P_2 будут выполняться в частях памяти одинакового размера в условиях одной и той же стратегии загрузки и замещения? Если ответ отрицательный, то как определяются размеры частей? Если положительный, то разумно ли сравнивать поведение различных программ, заставляя их умещаться в одинаковом объеме физической памяти? Это приемлемо, когда сравниваются две различные версии одной программы (см. разд. 9.4.2). Но в общем случае предпочтительнее индекс, отражающий только ссылочное поведение программы, присущее ей самой. Знание этого индекса и знание стратегии управления памятью при прохождении программы обеспечило бы идеальные условия для вычисления интенсивности отказов и, следовательно, времени выполнения программы.

Для большинства реальных задач полное описание ссылочного поведения программы [более точно, пары (программа, входные данные)] задается *трассировочной записью адресов* или *строкой ссылок* программы. Эту трассировочную запись можно получить из трассировочной записи для программы, введенной в разд. 9.2.1 (см. табл. 9.2); она дает ответы на все вопросы, касающиеся поведения программы. Предположим, как и прежде в этой главе, что виртуальная память выделяется программе статически. Другими словами, у каждого элемента информации свой точный виртуальный адрес, который

остается постоянным во все время жизни программы и не приписывается никакому другому элементу информации. Тогда существует взаимно-однозначное соответствие между словами программы и виртуальными адресами. Строка ссылок определяется как хронологическая последовательность виртуальных адресов a_j, на которые ссылается программа; с каждым адресом связан интервал времени ЦП (или виртуального) t_j до следующей ссылки:

$$\{a_j(t_j)\} \quad (j=1, 2, \ldots, r), \tag{9.4}$$

где r — общее число ссылок, генерируемых программой. Таким образом, строка ссылок — это одна из многих возможных декомпозиций времени выполнения, или, более точно, времени ЦП программы. Состояния, которые определяет эта декомпозиция, соответствуют отдельным адресам в занимаемом программой пространстве виртуальных адресов. Заметим, что это описание, производимое на уровне виртуального пространства и виртуального времени, полностью независимо от стратегии управления памятью.

Как и из трассировочной записи для программы (см. разд. 9.2.1), из строки ссылок можно вывести ряд дополнительных характеристик. Например, можно сгруппировать виртуальные адреса в блоки. Для заданного множества блоков $B=(b_1, b_2, \ldots, b_n)$ можно определить многозначное отображение множества виртуальных адресов A в B, такое, что каждый виртуальный адрес a_j связан с одним и только одним блоком b_j. Это отображение, примененное к каждому элементу трассировочной записи адресов (9.4), преобразует ее в *трассировочную запись блоков*, или *строку ссылок на блоки*:

$$\{i_j(t_j)\} \quad (j=1, 2, \ldots, r), \tag{9.5}$$

где для краткости b_{i_j} заменено на i_j.

Так как практический смысл введения блоков заключается в том, чтобы рассматривать их как отдельные единицы при переписях и операциях распределения, каждый блок обычно состоит из элементов информации, имеющих последовательные адреса в виртуальной памяти. Частные типы блоков — это *сегменты*, которые обычно являются логически и функционально хорошо определенными блоками переменной длины, и *страницы*, блоки фиксированной длины. Следовательно, в зависимости от типа рассматриваемого блока выражение (9.5) можно назвать *трассировочной записью сегментов* или *трассировочной записью страниц*. Обе эти трассировочные записи по-прежнему не зависят от стратегии и, как будет видно в примере 9.2, могут быть представлены в несколько более компактной форме, чем трассировочная запись адресов.

Но насколько можно сократить объем необходимой для представления поведения программы информации, сохранив при этом всю существенную информацию? Нельзя ли охарактеризовать поведение программы некоторой гистограммой, например, такой как профиль программы или смесь? Ответ, конечно, зависит от целей

нашего изучения. По мере увеличения зависимости модели поведения программы от стратегии уменьшается объем необходимой для описания производительности программы информации. В том случае, когда известно все, индекс может быть скаляром; например, это число страничных отказов. В другом случае, когда в описании не должно содержаться никакой информации о стратегии, нужно сохранить хронологический порядок ссылок и, по-видимому, нет достаточно удовлетворительного индекса, не имеющего форму последовательности. Некоторые из относительно простых моделей, приблизительно представляющих поведение программы для конкретных целей или для данных классов стратегий, будут описаны в разд. 9.3.2. Теперь мы обсудим пример, который позволит нам ввести фундаментальное понятие локальности.

Пример 9.2. Из трассировочной записи для программы в табл. 9.2 мы получим строку ссылок на блоки, представляющую ссылочное поведение программы из табл. 9.1. Предположим, что на первой карте ввода IENT=1, JNT=1 и на второй

Таблица 9.3

Декомпозиция на блоки программы из табл. 9.1

Номер блока	Содержимое блоков: Операторы	Содержимое блоков: Данные
1	MAIN(1–14), FLOAT	—
2	—	XO, YO, H, JNT, IENT, FUN, X, I, STEP, A(I)
3	FUN(12–14)	—
4	—	X, Y, FUN, 1.0
5	RK2(15–28)	—
6	—	H, XI, YI, K, N, H2, Y, X, FUN, II, J, T1, T2, T3, T4, VEC(II), 2.0, 6.0
7	READ	—
8	WRITE	—

IENT=0. Таким образом, рассматриваемая трассировочная запись состоит из первых 10 элементов первого столбца в табл. 9.2 и полного второго столбца, всего 43 элемента и 456 единиц времени. Мы произвольно подразделяем занимаемое программой виртуальное пространство на блоки переменной длины. Отображение операторов программы и данных в блоки приведено в табл. 9.3.

Читатель может удостовериться, что, зная это отображение и листинг программы из табл. 9.1, можно на основании нашей 43-элементной подзаписи трассировочной записи из табл. 9.2 вывести строку ссылок на блоки в табл. 9.4 (a). Эта строка построена для выполнения программы на машине ФОРТРАН, а не на реальной вычислительной машине. Каждый оператор выбирается из памяти и полностью декодируется. Затем производятся ссылки на все нужные данные, и оператор выполняется процессором ФОРТРАН без обращения к ссылочным операциям. Единственной уступкой реальному программированию является предположение, что операторы READ и WRITE реализованы двумя программами ввода-вывода, которые хранятся в блоках 7 и 8. Заметим, что строка, которая генерируется компилируемой программой, работающей на реальном ЦП, была бы значительно длиннее, с более короткими интервалами между последовательными ссылками. Эти более короткие интервалы настолько малы по отношению к интервалам, представляющим интерес при оценке производительности программ (например, время между страничными отказами или время ожидания стра-

ницы), что часто рассматривают лишь их среднюю длину, игнорируя таким образом всякое различие между ними. Для таких стратегий, как стратегия локального LRU-замещения или FIFO-замещения, как будет видно в разд. 9.3.1 (например, в табл. 9.8), времена ссылок не влияют на число страничных отказов и поэтому в трассировочной записи могут не учитываться. Отметим, что по своей точности строка из табл. 9.4 (a)

Таблица 9.4

Строка ссылок на блоки, генерируемая программой из табл. 9.1

(a) Строка ссылок на блоки

i_j	1	2	1	2	8	1	2	5	6	3	4	5	6	3	4	5	6	3	4	5	6
(l_j)	50	2	50	2	15	8	1	1	2	2	2	15	8	5	7						

i_j	5	6	3	4	3	5	6	7	5	6	3	4	3	5	6	7	5	6
(l_j)	11	15	8	5	7	11	15	8	5	7	4	15	8	5	7	23		

i_j	5	6	5	1	1	1	1	2	1	1	2	1	1
(l_j)	3	3	5	11	1	2	2	51	1	50	2	1	

(b) Строка множеств ссылок на блоки (одно множество каждые 25 единиц времени)

(1, 2, 7)(2, 7)(1, 2, 8)(1, 2, 8)(1, 2, 5, 6)(3, 4, 5, 6)(3, 4, 5, 6)(3, 4, 5, 6)(3, 4, 5, 6)
(3, 4)(3, 4, 5, 6)(3, 4, 5, 6)(5, 6)(1, 2, 5, 6)(1, 2, 8)(1, 2, 8)(1, 2, 7)(1, 2, 7)

пригодна для некоторых практических целей. Например, первые 50 единиц времени этой строки вместо последовательности 1 7 2 в строке объектной программы содержали бы более длинную последовательность различных сочетаний элементов 1, 7 и 2.

Другой формой для строки ссылок является форма с использованием *множеств ссылок*, показанная в табл. 9.4 (b). Для ее построения виртуальное время разделяется на интервалы постоянной длины. Перечень блоков, на которые делаются ссылки в течение каждого интервала, есть множество ссылок на блоки для этого интервала. Строка ссылок на множества менее точна, но более компактна, чем обычная строка ссылок. В дальнейшем будут показаны некоторые из ее применений.

Читатель может легко убедиться, что преобразование строки ссылок из табл. 9.4 (a) и (b) в строку ссылок на некоторые блоки является простой операцией, если существует многозначное отображение блоков из табл. 9.3 в новые блоки. Впрочем, это преобразование невозможно (по крайней мере когда требуется абсолютная точность) во всех других случаях. Если программа разделена на четыре страницы $\pi_1 = (b_1, b_3)$, $\pi_2 = (b_5)$, $\pi_3 = (b_7, b_8)$ и $\pi_4 = (b_2, b_4, b_6)$, то строка ссылок на страницы, соответствующая какой-либо строке из табл. 9.4, может быть без труда получена, если каждый номер блока заменить соответствующим номером страницы (например, 1 7 2 переходит в 1 3 4). Впрочем, если отображение многозначное в обе стороны, то требуется знание первоначальной трассировочной записи адресов, иначе придется довольствоваться неточным результатом. Например, если бы часть блока 2 входила в страницу π_4, а часть — в новую страницу π_5, то последовательность 1 7 2 могла бы преобразовываться в одну из последовательностей 1 3 4 или 1 3 5, или 1 3 4 5 в зависимости от того, на какие данные в блоке 2 фактически делает ссылки программа в этой точке.

Теперь строку из табл. 9.4 (a) проанализируем более детально. *Профиль блоков* программы для этого прогона, приведенный в табл. 9.5, показывает, что на блоки, как и на операторы, ссылаются по-разному. Как показано в табл. 9.5 различие в плотности ссылок может даже увеличиваться для более коротких интервалов. В течение любого произвольно выбранного интервала времени программа с большой вероятностью ограничится ссылками на некоторое подмножество своих блоков. Это можно легко увидеть на рис. 9.5, изображающем то, что мы будем называть *картиной* программы

[более точно, пары (программа, входные данные)]. В первые 100 единиц времени ссылки производятся только на блоки 1, 2, 7 и 8, а в следующие 30 единиц времени — только на блоки 5 и 6, являющиеся подмножеством множества блоков, которое никак не связано с предыдущим. Связь между этими подмножествами, которые называются *локальными областями*, может быть менее четкой, чем в нашем случае, так как у новой локальной области могут оказаться общие блоки со старой, но во времени она

Таблица 9.5

Плотности ссылок на блоки в строке из табл. 9.4(а)

Номер блока	Ссылки во всей строке (профиль блоков)	Ссылки в последних 20 элементах	Ссылки между 11-м и 50-м элементом
1	13	9	0
2	11	7	0
3	12	0	9
4	8	0	6
5	18	1	13
6	17	1	12
7	2	1	0
8	2	1	0

прослеживается совершенно отчетливо. В период выполнения программа находится в каждой локальной области l_j в течение времени t_j, которое можно рассматривать как *время жизни* области l_j. Поведение программы можно характеризовать последовательностью локальных областей и времен их жизни $\{l_j(t_j)\}$. Таким образом, когда требуется изобразить динамическое поведение программы, локальные области представляются естественной и удобной возможностью выбора состояний программы. Впрочем, идентификация локальных областей в строке ссылок на блоки не однозначна. Она зависит от выбранного масштаба времени. Например, две последовательности

Таблица 9.6

Два различных выбора локальных областей для строки ссылок на блоки из табл. 9.4

$\{l_j(t_j)\}$ = (1, 2, 7, 8)(104), (3, 4, 5, 6)(231), (1, 2, 7, 8)(121)

$\{l_j(t_j)\}$ = (1, 2, 7)(54), (1, 2, 8)(50), (5, 6)(31), (3, 4)(28), (5, 6)(18), (3, 4)(28), (5, 6)(18), (3, 4)(28), (5, 6)(11), (3, 4)(28), (5, 6)(41), (1, 2)(19), (1, 8)(50), (1, 2, 7)(52)

локальных областей в табл. 9.6 представляют собой одинаково законные характеристики поведения нашей программы, заданной табл. 9.4(а) и рис. 9.5. Проблема точного определения локальных областей будет обсуждаться в разд. 9.3.2.

Свойство объединения ссылок в относительно небольших областях всего виртуального пространства на сравнительно большом интервале виртуального времени называется *свойством локальности*. Каковы причины локальности? Читатель согласится, что для нашей программы локальность объясняется структурой кода и тем, как информация программы была разделена на блоки. Так как передача управления в программе, как правило, последовательная, и в течение даже относительно больших интервалов ссылки производятся лишь на небольшое подмножество данных, группировка соседних операторов и их данных в блоки приводит к тому, что у строки ссылок на блоки появляется в известной степени свойство локальности. Это явление проявляется сильнее в связи с циклами (которых нет в нашей строке, так как у нас ни один цикл не выполняется более одного раза) и массивами или другими структурами данных, которые программа обрабатывает в течение больших промежутков времени. За-

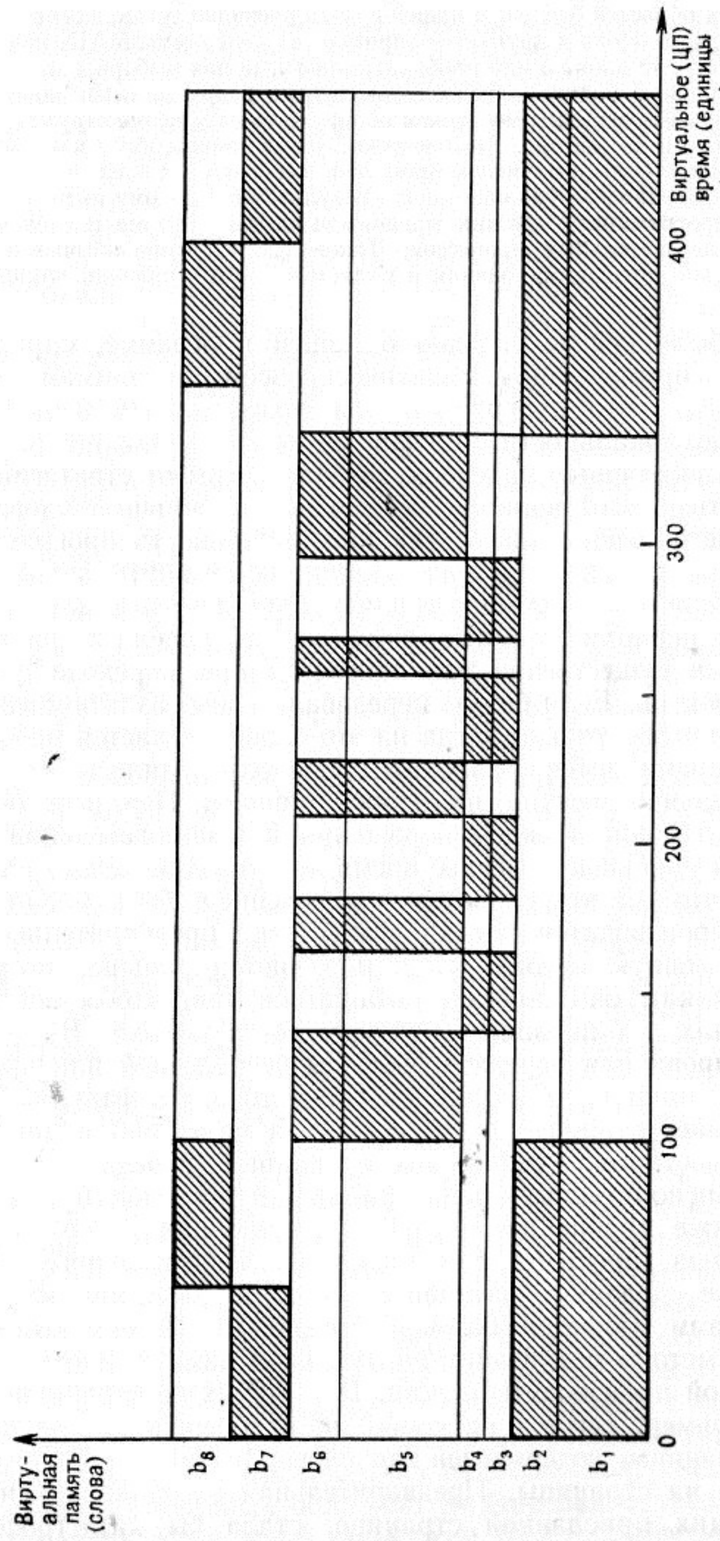


Рис. 9.5. Картина программы из табл. 9.1, полученная по строке ссылок на блоки из табл. 9.4 (а).

мены локальных областей бывают в нашей строке очевидно тогда, когда управление передается от одного блока к другому (например, из программы MAIN из блока 1 на подпрограмму RK2 из блока 5 или наоборот) или когда при выборе данных происходит переход от одного подмножества блоков данных в другое такое подмножество. В поведении нашей программы мы можем обнаружить отражение структуры мышления и способностей ее автора. Человеческий ум не приспособлен к одновременной работе со слишком большим числом процессов, особенно в деятельности, где желательна или существенна высокая надежность результатов. Поэтому интеллектуальная деятельность программиста должна организовываться в форме последовательных или почти последовательных процессов. Такое упорядочение обычно поощряется структурой языков программирования и является основой явления локальности. ■

То, что было сейчас сказано о нашей программе, справедливо, разумеется, в применении к практически всем программам. Эксперименты показали, что программы, как правило, обладают той или иной степенью локальности, но главный вопрос — можно ли эту степень оценить скалярным индексом, не зависящим от стратегии управления памятью. Этот вопрос, все еще не получивший удовлетворительного ответа, очень важен, так как локальность программы специфически используется при организации систем виртуальной памяти.

Виртуальная память обычно реализуется с помощью автоматически управляемой иерархии памяти, где, как мы видели в разд. 8.3.2, время доступа существенно увеличивается при переходе с первого на второй уровень. Если бы мы передавали слово из вспомогательной в основную память, только когда на это слово ссылается работающая программа, время выбора каждого слова определялось бы главным образом временем доступа на втором уровне. Передача большого количества слов при каждом обращении к вспомогательной памяти очень сильно уменьшает среднее время доступа для одного слова. Но ввиду того, что все эти слова, за исключением, быть может, слова, на которое производится ссылка, фактически предварительно загружаются в основную память (т. е. передаются раньше, чем на них делается ссылка), они должны выбираться так, чтобы вероятность ссылки на них в ближайшем будущем была высока. В противном случае массированная передача была бы бесполезной или вредной и не давала бы никаких преимуществ. Более того, тот факт, что в большинстве случаев второй уровень иерархии не является памятью со случайным доступом, делает массированные передачи удобными, только если используемые слова хранятся в последовательных ячейках. Это также крайне желательно в случае памяти со случайным доступом из-за простоты механизма последовательного доступа. Учитывая все сделанные замечания, легко понять, почему локальность программ является базовой предпосылкой, без которой существующие методы реализации виртуальной памяти не обеспечивали бы приемлемой производительности. В системах со страничной обработкой, например, ссылка на слово не из основной памяти влечет передачу страницы, содержащей это слово, из внешнего устройства, разделенного на страницы. Предварительная загрузка других слов, принадлежащих присланной странице, стала бы катастрофической

ошибкой, если бы вероятность того, что на большинство из них будут ссылаться в ближайшее время, равнялась бы нулю или была очень мала. Только тот факт, что программе присуще свойство локальности, дает возможность описанному механизму работать с приемлемой степенью эффективности.

Встав на другую точку зрения, можно сказать, что издержки времени, связанные с доступом к вспомогательной памяти, сводятся к приемлемому уровню, когда обеспечивается возможность хранения текущей локальной области целиком в основной памяти. Впрочем, если текущая локальная область всегда включает все страницы программы, мы приходим к системам с невиртуальной памятью или к находящимся в режиме постоянной толкотни системам с виртуальной памятью. Таким образом, локальность программы является существенным условием успешной работы систем с виртуальной памятью, эффективность которых увеличивается по мере усиления локальности (см. пример 7.10).

Рассмотрение всех этих моментов помогает нам понять, почему стратегии управления памятью используют при прогнозировании свойства локальности. Каждая стратегия пытается оценить состав текущей локальной области, чтобы хранить как можно большую часть ее в основной памяти. Естественно предположить, что программа будет оставаться в своей текущей локальной области (размер которой должен быть существенно меньше размера программы) достаточно долго, следовательно, и в ближайшем будущем. Другими словами, мы предполагаем, что программе в большой мере присуще свойство локальности. Однако различные стратегии основываются на различных определениях локальности. Например, стратегия фиксированного разбиения с замещением LRU предполагает, что в любой заданный момент вероятность ссылки на страницу максимальна для страницы, на которую сослались позднее всего, и минимальна для страницы, на которую сослались раньше всех других; поэтому хранение в основной памяти m позднее всего использованных страниц вероятнее всего минимизирует число страничных отказов. Следовательно, для этой стратегии локальность программы улучшается по мере того, как ее поведение становится более похожим на поведение, предполагаемое алгоритмом LRU. Естественно также ожидать, что производительность программы для данного выделенного пространства m пропорциональна соответствию ее поведения поведению только что описанной идеальной модели. Аналогично, стратегии рабочего множества оценивают размер и состав локальной области, уподобляя их размеру и составу рабочего множества, введенного в разд. 7.4.2. Следовательно, степень локальности программы (и ее производительность) в среде рабочего множества прямо связана с тем, насколько хорошо рабочее множество программы аппроксимирует ее текущую локальную область. Аппроксимация будет плохой, когда имеет место резкая смена локальных областей. Значительное увеличение активности аппарата страничной обработки будет соответствовать такой

смене не только для стратегии рабочего множества, но также для алгоритма LRU и для других стратегий, которые базируют свое прогнозирование на свойстве локальности.

Итак, мы показали, что локальность — это свойство, которое, грубо говоря, определяет производительность программы в среде с виртуальной памятью. Но удовлетворительная, точная, не зависящая от стратегии количественная мера локальности предложена не была. Хотя все стратегии управления памятью используют локальность, их определения локальности отличаются в той или иной степени. Отсюда следует, что от зависящих от стратегии индексов отказаться не так просто и что в ряде приложений их следует предпочесть не зависящим от стратегии характеристикам локальности. В любом случае для увеличения производительности программ и систем следует стремиться к разработке «локальных» программ и улучшать локальность существующих, особенно тогда, когда определение локальности для программы совпадает с ее определением для стратегии управления памятью, принятой в предполагаемой системе прохождения программы (см. разд. 9.4.2).

9.3. МЕТОДЫ ОЦЕНКИ ПРОГРАММ

9.3.1. Измерение программ

Введенные в разд. 9.2 индексы производительности программы не имели бы никакого смысла, если бы их нельзя было измерить экспериментально. Но как их можно измерить?

Измерение времени выполнения программы требует наличия часов и умения определять начало и конец прохождения программы. Как только становится известно определение этого индекса для конкретной среды и для требуемого исследования, эксперимент может быть разработан во всех деталях. Например, если определение совпадает с определением времени однопрограммного прохождения, можно пустить секундомер, когда программа начинает вводиться, и остановить его, когда все результаты полностью напечатаны. С другой стороны, эквивалент всех этих операций может выполняться самой должным образом инструментированной операционной системой. В большинстве случаев программа сбора статистических данных сообщает пользователю это время или некоторое его приближение. Если интерес представляет чистое время выполнения кода программы, в программу можно ввести две команды, считывающие показания внутренних часов системы, одна в начале и другая в конце кода. В примере 9.3 иллюстрируются некоторые применения этой элементарной техники, которую, естественно, можно использовать для измерения времени выполнения любой части программы.

Основные трудности при таком измерении времени обычно связаны с неточностью и недоступностью часов и с трудностью проверки того, что измеренные количества совпадают с интересующими. Фик-

сирующие реальное время часы, к которым можно обращаться только посредством привилегированных команд, т. е. только из операционной системы, часто недостаточны для измерения программного времени. Накладные расходы, связанные с частыми обращениями к часам, и особенно неточности, происходящие от нарушения естественного выполнения измеряемой программы, часто недопустимы в некоторых экспериментах. Кроме того, разрешающая способность часов должна отвечать требованиям эксперимента. Программа (или подпрограмма, или цикл), время выполнения которой равно примерно 10 мс, не должна пользоваться часами с разрешающей способностью 1/60 с. Использование часов, измеряющих реальное время, имеет тот недостаток, что невозможно провести различие между временем, потраченным на фактическое выполнение программы, и временем, истраченным, например, на обслуживание прерываний. Поэтому, если даже приняты все меры для полной изоляции от вмешательства процессов других пользователей, всегда нужно учитывать, что полученные данные сильно искажены непредсказуемыми и необнаруженными событиями. Часы, измеряющие виртуальное время, т. е. часы, которые работают в течение интервалов, когда выполняются индивидуальные процессы, безусловно, более надежны, хотя они обычно измеряют только время ЦП. В любом случае, как указано и в разд. 2.3.2, экспериментатор должен очень внимательно изучить характеристики имеющихся в системе часов, прежде чем инструментировать программу и разработать детали эксперимента. С точки зрения измерений программ пользователя крайне желательно наличие в системе часов с высокой разрешающей способностью и возможностью обращения к ним непривилегированными машинными командами; желательно также и наличие механизмов, с помощью которых операционная система может учитывать длительности определенных типов вмешательств в течение некоторых конкретных интервалов.

В простых случаях статического распределения памяти занимаемая программой память измеряется и результат обычно сообщается программисту загрузчиком, являющимся частью операционной системы. Таким образом, основные глобальные индексы производительности, введенные в разд. 9.2.1, можно измерять как на уровне операционной системы, так и на уровне кода, где программные средства измерения предпочтительнее аппаратных. Это замечание вполне можно отнести ко всем измерениям программ. В некоторых случаях можно использовать и средства машинного оборудования, но присущая им высокая разрешающая способность, как правило, не нужна, а отсутствие непосредственного контакта с измеряемой программой (для которой можно фиксировать только «электрические» проявления ее поведения) часто сильно ограничивает область их применения. Как отмечено в разд. 2.3.3, несколько по-другому обстоит в этом отношении дело с микропрограммами. Для них, наоборот, лучшими средствами часто являются средства микропрограммирования, концептуально подобные средствам программного обеспечения (разд. 2.4.1).

Поэтому мы ограничимся иллюстрацией использования при измерении программ средств программного обеспечения.

Мы уже говорили об использовании *контрольных точек* при измерении времени выполнения. В этом случае контрольные точки просто фиксируют начало и конец прохождения программы, считывают показания часов, запоминают результаты в специально зарезервированных ячейках. Технику контрольных точек можно использовать при всех измерениях программ, в которых каждое подлежащее обнаружению событие может быть однозначно связано с выполнением команды или оператора. Так как это, безусловно, включает и возможность трассировки программы, сформулированное в разд. 9.2.1 положение о трассировочной записи для программы означает, что контрольные точки позволяют получить ответы практически на все связанные с выполнением программы вопросы. Как будет видно из примера 9.3, чем более непосредственна связь между подлежащим обнаружению событием и выполнением команды, тем проще и дешевле измерение. Обращения к конкретным элементам данных или структур данных, которые могут выполняться различными командами и из различных программ или процедур, измерять гораздо труднее и дороже, чем обращение к конкретной подпрограмме. Аналогично, гораздо легче измерить, сколько раз выполняется определенный оператор, чем количество выполненных операций сложения. Трудности, о которых мы здесь говорим, не являются принципиальными. Например, если строится профиль программы, то определить количество операций сложения или число обращений к некоторой переменной в принципе не более сложно, чем определить число выполнений данного оператора, в чем можно убедиться на примере 9.1. Но если весь профиль нам не нужен и в нашем распоряжении его нет, то инструментирование программы с целью сбора упомянутых выше величин требует очень тщательного анализа текста программы и значительно большего числа контрольных точек. Стоимость оформления в программе контрольных точек и соответствующих измерительных подпрограмм для решения некоторой заданной задачи приблизительно можно считать пропорциональной числу внесенных контрольных точек. Это является и приемлемым первым приближением, когда используются средства автоматического внесения контрольных точек, такие, как Информатор и система SMT (см. разд. 2.4.2). Таким образом, хотя метод контрольных точек в принципе позволяет получить полную, детальную картину прохождения задачи, т. е. трассировочную запись для программы, обычно это делается, только когда абсолютно необходимо. На практике стараются обходиться минимальным или почти минимальным числом контрольных точек. Когда нужен профиль, программу не трассируют. Если желательно узнать число выполнений нескольких операторов, профиль программы излишен и поэтому, как правило, не измеряется. Как показывает пример 9.3, даже трассировка программы не требует по одной контрольной точке на команду или на оператор.

Пример 9.3. Мы опишем использование контрольных точек для инструментирования программы из табл. 9.1, так чтобы можно было экспериментально ответить на различные, связанные с производительностью вопросы относительно данной пары (программа, входные данные). В гл. 2 мы узнали, что концептуальная модель измеряемой системы лежит в основе любых измерительных экспериментов и что эта модель помогает экспериментатору при разработке опыта и интерпретации его результатов. В случае измерения программы методом контрольных точек естественнее всего подходят на эту роль *графовые модели*, полученные по блок-схеме программы [см., например, Раселл и Эстрин (1969)]. Блок-схема программы из табл. 9.1 приводится на рис. 9.6 (a). На рис. 9.6 (b) показана одна из многих возможных *графовых моделей*, которые можно получить по этой блок-схеме. При построении данного направленного графа сначала для каждого оператора образуется вершина. Такие операторы, как 5 или 21, которые содержат обращения к функции, должны быть представлены числом вершин, равным числу вызываемых ими функций плюс 1. Так например, оператор 5 разнесен на две вершины: первая соответствует действиям, имеющим место до обращения к функции FLOAT, а вторая — действиям, которые должны быть выполнены по возвращении из FLOAT. Далее вершины связываются друг с другом дугами, соответствующими возможным передачам управления. Графы, представляющие функции RK2 и FUN, вставлены в граф программы MAIN, так чтобы получился один связный граф. И наконец, все последовательно соединенные друг с другом вершины заменяются одной. Говорят, что вершины n_i и n_j последовательно соединены, если существует дуга из n_i в n_j и она является единственной выходящей дугой для n_i и единственной входящей дугой для n_j. Смысл этого упрощения очень прост и очень важен: две последовательно соединенные вершины всегда выполняются одинаковое число раз, если только выполнение программы не приостанавливается или она не выбрасывается во время выполнения первой вершины из-за ошибки или по приказу оператора. Понятие последовательного соединения можно обобщить на подграфы, что, безусловно, знают читатели, знакомые с графовыми моделями электрических цепей и законами Кирхгофа. Если связный подграф имеет только одну вершину с входящей дугой из остальной части графа и только одну вершину (не обязательно другую) с выходящей дугой в остальную часть графа, то данный подграф можно заменить одной единственной вершиной, и если эта вершина последовательно соединена с другими, то их можно объединить. Например, подграф, состоящий из вершины VII и цикла на рис. 9.6 (b), удовлетворяет сформулированному выше условию. Если мы заменим его одной вершиной, то увидим, что эта вершина будет последовательно соединена с вершиной VI. Обе эти вершины всегда выполняются одно и то же число раз. Аналогично, можно прийти к заключению, что III и IV выполняются одинаковое число раз и что число выполнений вершины I равно числу выполнений II и VI плюс 1. Впрочем, необходимо отметить, что сведение графовой модели должно учитывать глобальные цели анализа. Сведение, в результате которого получилась графовая модель на рис. 9.6 (b), отвечает нашим требованиям, в чем мы убедимся ниже. Описанные выше дополнительные сведения, хотя и дают нам возможность уменьшить число контрольных точек для некоторых задач, могут настолько упростить модель, что мы не сможем измерить, например, число инициируемых программой операций WRITE, так как становится невозможным измерить число выполнений вершины VII из-за исчезновения цикла.

Теперь подробно исследуем инструментирование, которое позволит нам ответить на некоторые конкретные вопросы о производительности нашей программы.

a) Каково время выполнения? В программу нужно вставить две контрольные точки на дугах с метками a, как показано на рис. 9.6 (b). Соответствующая модификация кода в предположении существования в библиотеке ФОРТРАНа программы ITIME (K) приводится в табл. 9.7 (a). При обращении программа ITIME (K) выдает текущее значение реального времени в форме целого. Если у измеряемой программы несколько выходов, может оказаться удобным преобразовать ее в программу с одним выходом прежде, чем вставлять контрольные точки.

b) Сколько времени использует подпрограмма RK2? Требуемое инструментирование подобно предыдущему с учетом того, что подпрограмма может выполняться несколько раз за время прохождения программы (см. табл. 9.7 (b) и метки b на рис. 9.6 (b)). Заметим, что мы предполагаем, что время, затрачиваемое функцией

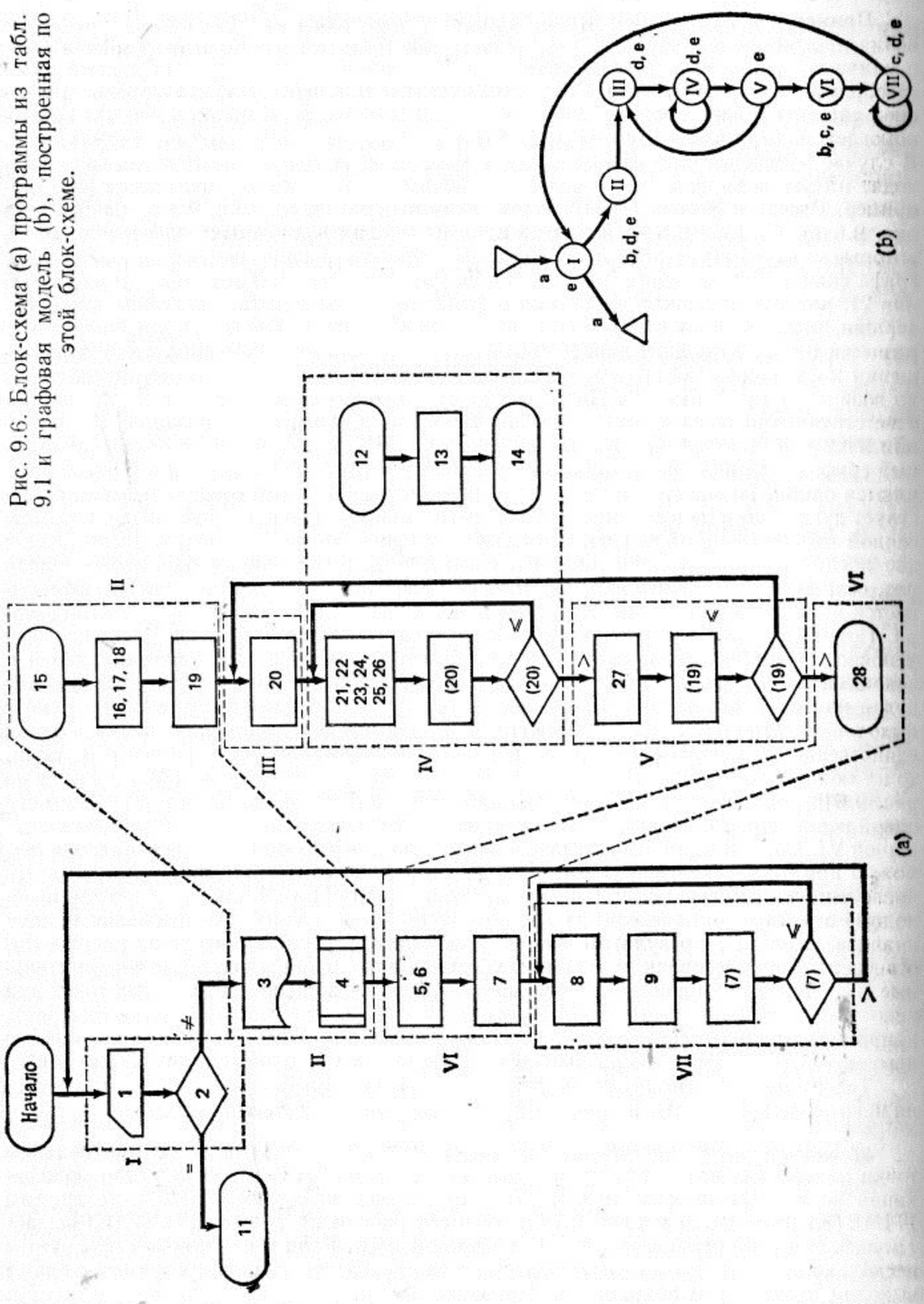


Рис. 9.6. Блок-схема (a) программы из табл. 9.1 и графовая модель (b), построенная по этой блок-схеме.

FUN, является частью времени, которое мы хотим измерить. В задаче 9.6 читателя просят так инструментировать эту программу, чтобы соответствующие времена измерялись раздельно.

c) Сколько раз обращаются к переменной STEP? Эта переменная встречается в операторах 5 и 8 по одному разу каждая. Таким образом, эта проблема эквивалентна проблеме определения того, сколько раз выполняются операторы 5 и 8. Достаточно вставить две контрольные точки, исполняющие роль счетчиков, там, где на рис. 9.6 (b) дуги помечены меткой c. Соответствующая модификация приведена в табл. 9.7 (c).

Таблица 9.7

Модификации, инструментирующие программу из табл. 9.1

Номер оператора		Новый оператор
(a) Измерение времени выполнения		
1		CALL ITIME(INIT)
1′	10	READ(5,1) XO,YO,H,JNT,IENT
11	40	CALL ITIME(IFIN)
11′		KTIME = IFIN − INIT
11″		WRITE(6,4) KTIME
11‴		STOP
(b) Измерение времени для подпрограммы RK2		
1′		KTRK2 = 0
4		CALL RK2(FUN,H,XO,YO,JNT,IENT,A,KTRK2)
11	40	WRITE(6,5) KTRK2
11′		STOP
15		SUBROUTINE RK2(FUN,H,XI,YI,K,N,VEC,KTRK2)
15′		CALL ITIME(INIT)
28		CALL ITIME(IFIN)
29		KTRK2 = KTRK2 + IFIN − INIT
30		RETURN
(c) Измерение числа обращений к переменной STEP		
1		N58 = 0
1′	10	READ(5,1) XO,YO,H,JNT,IENT
5′		N58 = N58 + 1
8′		N58 = N58 + 1
11	40	WRITE(6,6) N58
11′		STOP
(d) Измерение профиля программы		
1		NII = 0
1′		NIII = 0
1″		NIV = 0
1‴		NVII = 0
1ᴵᵛ	10	READ(5,1) XO,YO,H,JNT,IENT
3′		NII = NII + 1
7′		NVII = NVII + 1
11	40	WRITE(6,7) NII,NIII,NIV,NVII
11′		STOP
19′		NIII = NIII + 1
20′		NIV = NIV + 1
(e) Построение трассировочной записи для программы		
1		CALL TRACE(0)
1′	10	CALL TRACE(1)
1″		READ(5,1) XO,YO,H,JNT,IENT
3	20	CALL TRACE(2)
3′		WRITE(6,2) H,XO,YO
7′		CALL TRACE(7)
19′		CALL TRACE(3)
20′		CALL TRACE(4)
26′		CALL TRACE(5)
27′		CALL TRACE(6)

Продолжение

Номер оператора	Новый оператор
Программа измерения	
29	SUBROUTINE TRACE(J)
30	IF (J.EQ. 0) L = 0
31	L = L + 1
32	LTRAC(L) = J
33	CALL ITIME(K)
34	LTIME(L) = K
35	RETURN
	END

d) Каков профиль программы? Учитывая специфику построения графовой модели на рис. 9.6 (b), мы знаем, что операторы, которым соответствует одна и та же вершина, всегда выполняются одинаковое число раз. Так обстоит дело, например, с операторами 3, 4, 15, 16, 17, 18 и 19, которые будут выполняться NII раз. Единственным исключением являются операторы 12, 13 и 14, каждый из которых выполняется 4·NIV раз. Исследование графовой модели показывает также, что в этом примере, как было отмечено раньше, справедливы следующие соотношения между числом выполнений ее семи вершин:

$$NII=NVI,\ NIII=NV,\ NI=NII+1. \tag{9.6}$$

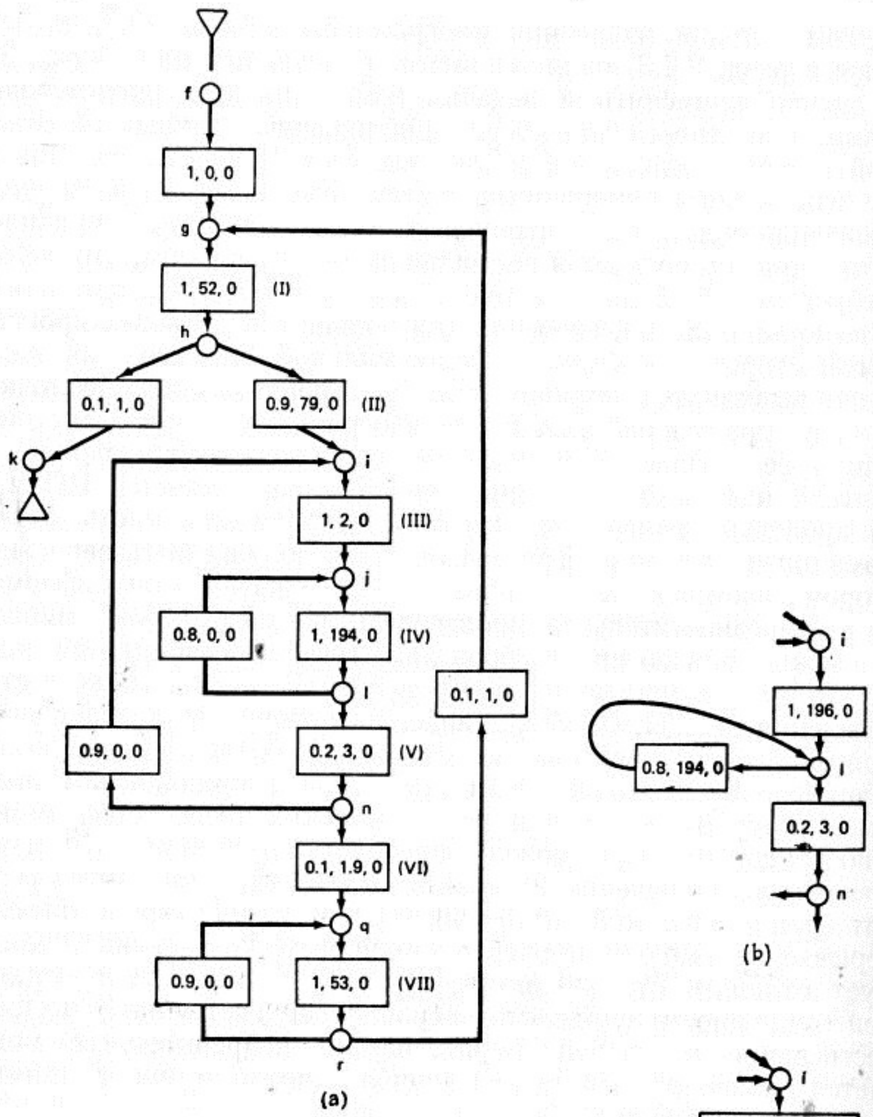
Из этих чисел только четыре (например, NII, NIII, NIV и NVII) независимы, и весь профиль можно получить с помощью только четырех контрольных точек [см. табл. 9.7 (d)]. Заметим, что, так как профиль является и мощным средством отладки, внесение избыточных контрольных точек может оказаться полезным для повышения основательности и надежности тестирования. Иногда, особенно в случае больших программ, может оказаться более удобным организовать все счетчики в виде вектора и превратить каждую контрольную точку в обращение к подпрограмме, увеличивающей один из элементов этого вектора.

e) По какому пути идет передача управления? Трассировочную запись для программы в терминах вершин графовой модели на рис. 9.6 (b) можно получить, если в каждую вершину ввести контрольную точку трассировочного типа. И здесь мы могли бы сократить общее число контрольных точек, если бы стали учитывать, например, тот факт, что вершина VI всегда выполняется между выполнением вершин V и VII. Поэтому вершину VI трассировать не обязательно. Аналогичные соображения показывают, что не нужно трассировать и вершины I и VI. Поэтому было бы достаточно только четырех контрольных точек, но тогда обработка такой экономной трассировочной записи стала бы занимать больше времени и обошлась бы дороже, хотя сама запись оказалась бы дешевле, а уровень искажений был бы ниже. Кроме того, когда программа трассируется с целью отладки, часто разумнее оказывается использовать большее число контрольных точек, чем строгий минимум. В табл. 9.7 (e) представлены все семь контрольных точек, необходимых для построения полной трассировочной записи, и текст программы измерения TRACE. Дополнительная контрольная точка, выполняемая только один раз, CALL TRACE (0) используется для инициализации указателя на буфер этой программы LTRAC. Код программы TRACE содержит считывание и запись времени (операторы 33 и 34). Это может оказаться необязательным и, по всей видимости, приведет к значительным дополнительным потерям времени и искажениям при работе процедуры трассировки. Чтобы свести искажения к минимуму, в таких случаях нужно использовать контрольные точки, которые увеличивают указатель L и записывают номер блока в LTRAC (L) локально, без обращения к каким-либо подпрограммам. При необходимости операция считывания показаний часов должна быть введена как машинная команда, при условии что она предусмотрена в ЦП. Векторы LTRAC и LTIME следует хранить в области COMMON. Для краткости в текст программы TRACE в табл. 9.7 (e) не включена обработка буфера переполнения. ■

Другим методом измерения программного обеспечения, который вводился в разд. 2.4.3, является метод, основанный на *выборке*. Этот метод можно применять в задачах измерения производительности программ, в частности для сбора приближенных данных об относительной частоте выполнения различных частей программы. Как и в случае измерения операционных систем, его главное преимущество по сравнению с методом контрольных точек состоит, как правило, в том, что меньше нарушается нормальная работа, и в том, что выборка не требует модификации измеряемой программы. Последнее обстоятельство крайне важно, особенно для больших и сложных программ. С другой стороны, выборка — менее сильное средство, так как не позволяет выполнять измерения типа трассировки, и она менее точна.

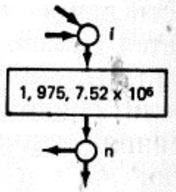
Если программа работает в однопрограммном режиме, мы можем оценивать ее активность с помощью часов реального времени или измерителя интервалов времени, периодически прерывая ЦП и вызывая процедуру измерения. Эта процедура должна иметь доступ к информации о статусе программы, которая должна быть записана на некотором запоминающем устройстве. При сжатии этой информации (которое предпочтительнее производить позднее, чтобы минимизировать искажения во время сбора данных) она позволит нам определить, что делала программа в это время. Если она была в стадии прохождения на ЦП, то по команде, на которую указывает счетчик программы, программа сжатия определит область кода, который выполнялся. Более точно, если эта команда принадлежит подпрограмме, которая вызывается из нескольких точек программы, то обычно трудно или невозможно определить, из которой точки к ней производилось обращение, и область активности приходится ограничивать границами самой подпрограммы. Так же обстоит дело в случае с прерываемыми модулями операционной системы, активность которых следует отличать от активности программы, так как программист лишен возможности управлять операционной системой. Заметим, что существование в операционной системе непрерываемых модулей является одним из источников ошибок, которые могут влиять на результаты, получаемые методом выборки.

Если обнаруживается, что программа находилась в состоянии ожидания, записанная информация о статусе позволит программе сжатия данных определить причину, по которой программа ждет. В мультипрограммной среде программа может ждать, вероятнее всего, того момента, когда станут доступными необходимые ей ресурсы, а не завершения операций ввода-вывода, которые она явно затребовала. Эти два типа ожидания следует рассматривать раздельно, так как программист в какой-то мере может влиять на второе, но никак не на первое. Если программа измерения имеет доступ к базе данных операционной системы, этой информацией можно пользоваться и ее можно запомнить. Чтобы иметь возможность это сделать, программа должна работать в привилегированном режиме и фактически быть частью операционной системы. Если выборочный измеритель должен выпол-

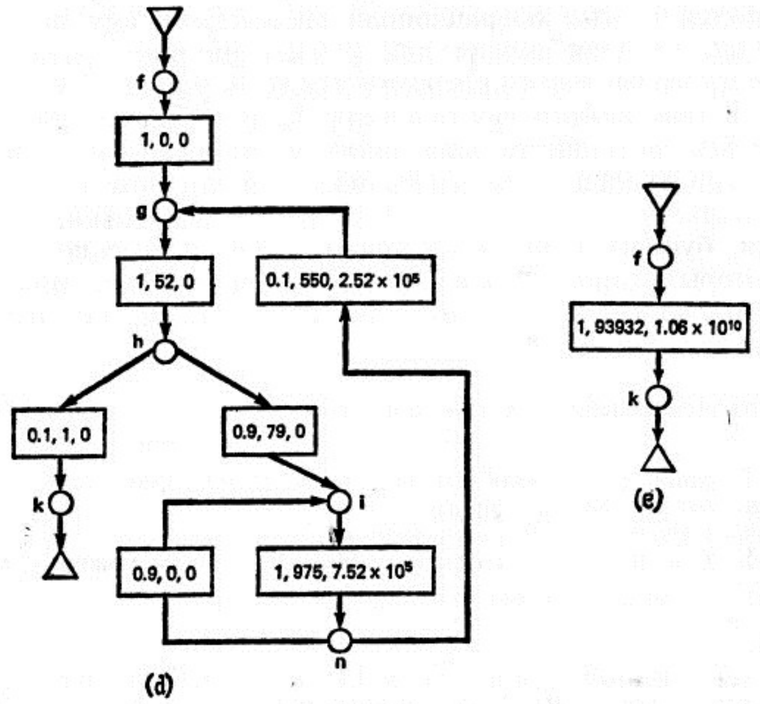


(a)

(b)



(c)



(d)

(e)

Рис. 9.7. Вычисление времени выполнения программы из табл. 9.1 посредством последовательного исключения вершин в соответствии с методом, предложенным Бейзером (1970).

няться как работа пользователя и не требовать никакой модификации операционной системы, то такая задача, как правило, очень трудна и даже может оказаться неразрешимой. В измерителе проблемных программ PPE, разработанном корпорацией «Бул и Бэббидж» для программ, работающих на системах IBM/360 и 370, эта трудность могла быть решена для ожиданий ЦП (т. е. времен готовности) только за счет некоторых неиспользуемых битов в векторе статуса программы [см. Холтвик (1971)]. Однако ожидание в очереди для других ресурсов оказалось невозможным отделить от времени, когда эти ресурсы использовались фактически. Пользователю выдавалось общее время для каждого ресурса, которое PPE должным образом определяет в логических, а не физических терминах, т. е. в терминах множеств данных, а не элементов оборудования. Идеально выборка в программе должна делаться не в реальном времени, а в виртуальном (ЦП), если мы хотим получить профиль программы, или на базе однопрограммного времени выполнения, если нас интересует также производительность ее операций ввода-вывода.

В системах с динамическим распределением памяти и виртуальной памятью метод проб не может выполняться на уровне физической памяти, так как в разное время абсолютные адреса указывают на различные элементы информации. Однако если выборочный измери-

тель является частью операционной системы, то ему доступна вся информация, которая нужна для работы на виртуальном уровне. Метод контрольных точек пригоден в таких ситуациях так же хорошо, как и в системе любого другого типа. Единственной проблемой, которая может возникнуть для инструментированной программы в системе с виртуальной памятью, является то, что иногда трудно оценить влияние инструментации (контрольные точки, программы измерения, буферы данных) на динамическое поведение программы. При некоторых типах измерения это влияние может оказаться неожиданно высоким, а снизить его может оказаться довольно трудно.

Таблица 9.8

Алгоритм моделирования стратегии локального замещения LRU

Входы: l = длина исследуемой строки ссылок на страницы
u = число страниц в программе
m = выделенная программе память
$\{i_j\}$ = строка ссылок на страницы (времена ссылок можно не учитывать)

Выходы: nf = число генерируемых страничных отказов

Алгоритм:

1. (Присвоить начальные значения стеку LRU, указателю на текущую ссылку, числу страничных отказов.) Установить $s_h \leftarrow 0$ для $h=1, \ldots, u$, $j \leftarrow 0$, $nf \leftarrow 0$.
2. (Перейти к следующей ссылке на страницу и к вершине стека.) Установить $j \leftarrow j+1$, $k \leftarrow 1$.
3. (Найти в стеке текущую ссылку.) Если $s_k = 0$, установить $\delta \leftarrow \infty$ и перейти к 4. Если $s_k = i_j$, установить $\delta \leftarrow k$ и перейти к 4. В противном случае установить $k \leftarrow k+1$ и повторить этот шаг.
4. (Определить, генерируется ли страничный отказ.) Если $\delta > m$, установить $nf \leftarrow nf+1$.
5. (Обновить стек.) Установить $s_h \leftarrow s_{h-1}$ для $h=k, k-1, \ldots, 2$, $s_1 \leftarrow i_j$.
6. (Проверить условие окончания.) Если $j < l$, перейти к 2. В противном случае окончание.

Измерение зависящих от стратегии индексов поведения программы, которые рассматривались в разд. 9.2.2, может выполняться обычно только операционной системой. Например, никакая контрольная точка в программе пользователя не может обнаружить страничные отказы. Другой путь выполнения этих измерений состоит в моделировании стратегии управления памятью и введении в модель трассировочной записи адресов, генерируемой программой при одном из ее прохождений [см., например, Биледи (1966)]. Такую управляемую трассировочной записью модель нетрудно разработать и реализовать. В табл. 9.8 приведен алгоритм определения числа страничных отказов, генерируемых строкой ссылок на страницы при стратегии замещения LRU. В табл. 9.9 показан алгоритм вычисления числа страничных отказов и среднего размера рабочего множества программы при простой стратегии рабочего множества. Тем читателям,

которые никогда этого не делали, мы советуем смоделировать обе эти стратегии вручную, применив алгоритмы из табл. 9.8 и 9.9 к какой-нибудь короткой строке, например к строке из табл. 9.4(а) [см. задачи 9.8 и 9.9].

Из табл. 9.8 легко увидеть, что страничный отказ генерируется тогда, когда δ, расстояние текущей ссылки от вершины стека LRU,

Таблица 9.9

Алгоритм моделирования простой стратегии рабочего множества

Входы: l = длина исследуемой строки ссылок на страницы
u = число страниц в программе
$\{i_j\}$ = строка ссылок на страницы (предполагается, что ссылки генерируются через одинаковые интервалы, поэтому их времена не учитываются)
τ = размер окна (в числе ссылок)

Выходы: nf = число генерируемых страничных отказов
$\bar{w}$ = средний размер рабочего множества

Алгоритм:

1. (Присвоить начальные значения сумме размеров рабочих множеств, указателю на текущую ссылку, указателю на самую последнюю ссылку в окне, числу страничных отказов, счетчикам страниц в окне.) Установить $sw \leftarrow 1$, $j \leftarrow 1$, $k \leftarrow 1$, $nf \leftarrow 1$, $c_h \leftarrow 0$ для $h=1, \ldots, u$, $c_{i1} \leftarrow 1$.
2. (Перейти к следующей ссылке и определить, генерируется ли страничный отказ.) Установить $j \leftarrow j+1$. Если $c_{ij}=0$, установить $nf \leftarrow nf+1$.
3. (Обновить счетчики u, а также при необходимости окно.) Установить $c_{ij} \leftarrow c_{ij}+1$. Если $j-k+1 > \tau$, установить $k \leftarrow k+1$, $c_{ik} \leftarrow c_{ik}-1$.
4. (Вычислить размер рабочего множества и обновить сумму.) Установить $s \leftarrow 0$. Для $h=1, \ldots, u$, если $c_h > 0$, установить $s \leftarrow s+1$. Установить $sw \leftarrow sw+s$.
5. (Проверить условие окончания.) Если $j < l$, перейти к 2.
6. (Вычислить средний размер рабочего множества). Установить $\bar{w} \leftarrow sw/l$. Окончание.

больше размера отведенной памяти m. Стек LRU от m не зависит, так как он содержит все ссылки на страницы, которые встречались ранее, безотносительно к тому, сколько памяти было отведено программе. Если вместо того, чтобы специфицировать m, мы заведем счетчики, по одному на каждую позицию стека, и при каждой ссылке будем увеличивать все соответствующие позиции стека, для которых $h<\delta$, то в каждый момент h-й счетчик содержит число страничных отказов, которые генерировались бы к этому моменту, если бы m равнялась h страницам. Это наблюдение, являющееся основой *метода стековой обработки* [см. Маттсон и др. (1970)], позволяет конструировать кривую времени жизни для пары (программа, входные данные) при стратегии LRU за один просмотр строки ссылок (см. задачу 9.8). Заметим также, что времена обращений к страницам можно не учитывать, так как число страничных отказов от них не зависит.

Когда исследоваться должна только часть трассировочной записи адресов, измерение упомянутых выше индексов бессмысленно, если трассировочная запись очень короткая. Имеющаяся литература в этой области показывает, что используемая при таком имитационном моделировании длина трассировочной записи обычно порядка 10^6 ссылок, что соответствует времени ЦП порядка 1 с (конечно, это зависит от скорости ЦП и полосы частот основной памяти). Стоимость описанного моделирования, как правило, достаточно высока, если только тип стратегии или требования точности не позволяют экспериментатору сократить степень детализации трассировочной записи. Например, при работе с некоторыми стратегиями рабочего множества строка ссылок на множества [см. табл. 9.4(b)] часто достаточно точна, как мы увидим ниже. Стоит отметить, что имитация позволяет измерять индексы поведения программы в очень хорошо управляемой среде, тогда как непосредственное измерение тех же самых индексов часто подвержено влиянию таких факторов, как текущая нагрузка системы и поведение остальных программ. Таким образом, когда целью является оценка программ, имитация предпочтительнее. В задачах оценки стратегий непосредственные измерения или по крайней мере имитация мультипрограммной среды многократными трассировочными записями имеют свои преимущества, и мы советуем выбирать их.

Как можно провести трассировку адресов? В разд. 9.2.2 мы видели, что трассировочную запись адресов, являющуюся полной характеристикой пары (программа, исходные данные), можно вывести из трассировочной записи программы на уровне машинного языка, если виртуальный адрес каждой команды и каждого элемента данных программы известны и если специфицирован механизм доступа к ним (например, косвенная адресация, индексирование). Впрочем, объем информации, который должен сопутствовать следу программы для обеспечения вывода соответствующей трассировочной записи адресов, настолько велик, что этот метод применяется редко, за исключением того случая, когда достаточна приблизительная трассировочная запись, полученная при некоторых упрощающих предположениях.

Более простой, хотя и довольно дорогой, способ получения точной трассировочной записи адресов основывается на *интерпретационном выполнении* программы. При этом способе программа на машинном языке передается должным образом инструментированному интерпретатору, который выполняет ее, моделируя реальную систему. Интерпретатор инструментируется так, чтобы фиксировать все ссылки на память, которые делает программа, а также при необходимости (моделируемые) времена этих ссылок. Следует заметить, что главной целью интерпретационного выполнения является замедление обработки программы, так чтобы имелось время для фиксирования ссылочных адресов. В конечном счете команды программы в течение нормального выполнения интерпретируются аппаратурой управления ЦП. Можно сконструировать аппаратуру так, чтобы она обеспечивала выделение генерируемых ЦП адресов, запись их в буфер и периодическую перепись на некоторое вспомогательное запоминающее устройство. Фактически некоторые из аппаратных средств, описанных в разд. 2.3.4, можно использовать для сбора элементов трассировки адресов в реальном времени. Можно представить себе и специально встроенные программные средства, помогающие решению этой задачи. Однако недоступность или высокая стоимость этих средств сделали интерпретационное выполнение самым распространенным методом измерений этого типа, несмотря на издержки по разработке интерпретатора и расходу времени вычислительной машины (обычным нужно считать случай, когда отношение реального времени к моделируемому достигает величины нескольких порядков).

9.3.2. Моделирование программ

Предложено и используется много типов моделей, представляющих программы для вычислительных машин. В различных приложениях часто требуются различные типы моделей программ. Мы уже приводили некоторые примеры. В гл. 5 мы видели, что при изучении оценки систем программы часто представляются тем, как они запрашивают ресурсы, и что имеются различные уровни ресурсов. Смесь команд или операторов, описанная в разд. 5.3.4, и модель ресурсов языка высокого уровня, используемая в разд. 6.3.4, как раз являются двумя типами моделей этого класса. В разд. 9.3.1 графовые модели оказались самым подходящим концептуальным представлением программ для целей измерения. Еще больше моделей было придумано для проблем, не так тесно связанных с оценкой производительности: проверка правильности, обнаружение параллелизма, трансляция программы и другие.

В этом разделе мы ограничимся рассмотрением нескольких типов моделей, которые можно использовать для анализа производительности программ, т. е. для оценки значений некоторых из индексов, введенных в разд. 9.2.1 и 9.2.2. В частности, в первой части этого раздела мы будем заниматься оценкой времени выполнения и оценкой профиля программы как аналитическими методами, так и с помощью имитации, а во второй части — некоторыми простыми моделями поведения программы.

Первой мы будем рассматривать *марковскую модель*, предложенную Бейзером (1970) для аналитического определения времени выполнения программы. Как мы увидим далее в этом разделе, марковские процессы были и являются весьма популярными при моделировании программы [см., например, Рамамурти (1965)]. Модель Бейзера представляет собой ориентированный граф, в котором каждая дуга характеризуется тройкой (p, u, v), где p — условная вероятность передачи управления по данной дуге при достижении исходной вершины дуги; u — среднее время выполнения дуги; а v — дисперсия этого времени. Распределение времен выполнения для каждой дуги произвольное; единственно, что требуется, это существование среднего

времени и его дисперсии. Ниже метод Бейзера иллюстрируется его применением к программе из табл. 9.1.

Пример 9.4. Графовая модель нашей программы, показанная на рис. 9.6 (b), может быть преобразована в модель на рис. 9.7 (a). В этом случае вершины представляют соединения и точки выбора, а дуги представляют блоки операторов, которые последовательно связаны друг с другом (см. определение последовательной связности в примере 9.3). Таким образом, в этой модели вся обработка сконцентрирована в дугах, тогда как в модели на рис. 9.6 (b) — в вершинах. У каждой дуги имеется метка, которая на рис. 9.7 (a) заключена в прямоугольник. Эта метка является тройкой (p, u, v), которая была введена выше. Значения среднего времени выполнения для дуги выведены по значениям времени из табл. 9.2. Предполагается, что во всех случаях дисперсия равна нулю. Фактически изменчивость времен выполнения большинства

Таблица 9.10

Уравнения для исключения вершин в марковской модели программы

Последовательные преобразования: $a \to b \to c \Rightarrow a \to c$
$p'_{ac} = p_{ab}p_{bc}$
$u'_{ac} = u_{ab} + u_{bc}$
$v'_{ac} = v_{ab} + v_{bc}$
Параллельные преобразования: $a \overset{\circ}{\underset{\circ}{\rightrightarrows}} b \Rightarrow a \to b$
$p'_{ab} = p^{\circ}_{ab} + p^{\circ\circ}_{ab}$
$u'_{ab} = (p^{\circ}_{ab}u^{\circ}_{ab} + p^{\circ\circ}_{ab}u^{\circ\circ}_{ab})/(p^{\circ}_{ab} + p^{\circ\circ}_{ab})$
$v'_{ab} = (p^{\circ}_{ab}v^{\circ}_{ab} + p^{\circ\circ}_{ab}v^{\circ\circ}_{ab})/(p^{\circ}_{ab} + p^{\circ\circ}_{ab}) + [(u^{\circ 2}_{ab}p^{\circ}_{ab} + u^{\circ\circ 2}_{ab}p^{\circ\circ}_{ab})/(p^{\circ}_{ab} + p^{\circ\circ}_{ab})] - u'^2_{ab}$
Преобразования цикла: $\circlearrowleft a \to b \Rightarrow a \to b$
$p'_{ab} = p_{ab}/(1 - p_{aa})$
$u'_{ab} = u_{ab} + p_{aa}u_{aa}/(1 - p_{aa})$
$v'_{ab} = v_{ab} + [v_{aa}p_{aa}/(1 - p_{aa})] + u^2_{aa}p_{aa}/(1 - p_{aa})^2$

Источник: Бейзер (1970)

команд на большинстве машин так мала, что ею можно пренебречь. Вообще говоря, это неверно для таких дуг, как gh, hi и gr, которые содержат операторы READ и WRITE. Однако для простоты мы также предполагаем, что дисперсия времени их выполнения равна нулю. Самой трудной задачей является обычно оценка вероятностей. Это вероятности перехода от одной вершины к другой. Предполагается, что они постоянны, т. е. не зависят от времени и от пути, по которому шли передачи управления до исходной вершины. Последнее предположение, делающее эту модель марковской, особенно нереалистично, так как оно, безусловно, не верно для большинства команд условной передачи управления в практических программах. Бейзер (1970) классифицирует условные передачи на следующие четыре категории:

a) те разветвления, условия, или *предикаты*, которых зависят только от исходных данных. Если значения исходных данных известны, то вероятности переходов легко могут быть вычислены по этим значениям. Например, если оператор 2 нашей программы записан в виде IF (IENT.EQ.0) GOTO 40, то его предикатом, естественно, является (IENT=0). Следовательно, оценка вероятностей p_{hk} и p_{hi} эквивалентна

оценке среднего числа значимых вводимых карт на один прогон, т. е. числа карт, у которых IENT≠0. Если это число равно 10, то $p_{hk}=0.1$ и $p_{hi}=0.9$;

b) те разветвления, предикаты которых зависят от структуры программы, но вероятность которых легко оценить из-за простоты этой структуры и соответствующих зависимостей. Примером является разветвление в конце цикла DO, состоящего из операторов 7, 8 и 9. Его предикат имеет вид (I⩽IENT); изучение данного оператора DO позволяет сделать вывод, что если IENT⩾1, то $p_{rg}=1/\mathrm{IENT}$ и $p_{rq}=(\mathrm{IENT}-1)/\mathrm{IENT}$. Поэтому, если мы ожидаем, что IENT примерно равно 10, мы получаем значения, приведенные на рис. 9.7 (a), для этих двух вероятностей и для всех вероятностей такого типа;

c) те разветвления, вероятность которых не имеет существенного значения для вычисления времени выполнения. Например, если среднее время выполнения двух параллельных ветвей примерно одинаково, то результирующее эквивалентное время выполнения существенно не зависит от их индивидуальных вероятностей (см. уравнение u'_{ab} для параллельного преобразования в табл. 9.10);

d) те разветвления, вероятность которых оценить трудно главным образом из-за того, что их придикаты связаны со структурой программы сложной зависимостью, или из-за сложности самой структуры, или из-за того и другого.

Бейзер утверждает, что только 5% ветвей принадлежат категории d и что быстродействие его метода, который легко автоматизируем, позволяет исследовать чувствительность общего времени выполнения к значениям относящихся к таким ветвям вероятностей, изменяя их и повторяя анализ. При этом могло бы оказаться возможным сократить или даже исключить трудоемкий анализ, который потребовался бы для оценки вероятностей таких разветвлений.

Как только графовая модель построена, оценка времени выполнения может быть выполнена методом *исключающих преобразований*, который хорошо известен в теории электрических цепей. Этот метод состоит из следующих шагов.

Шаг 1. Выбрать вершину, не являющуюся входом или выходом, с целью ее исключения. Например, пусть это будет вершина j на рис. 9.7 (a).

Шаг 2. Размножить выбранную вершину на копии, так чтобы ее выходящие дуги были последовательно связаны с заходящими дугами. Например, j размножена на две вершины j' и j'', так что дуги lj' и $j'l$ последовательно связаны; это же верно и для дуг ij'' и $j''l$. Новые дуги $j'l$ и $j''l$ имеют одинаковые метки с меткой jl.

Шаг 3. Применить уравнения последовательного преобразования из табл. 9.10 к только что созданным последовательным связям с целью исключения новых вершин. В нашем примере мы получаем две дуги il и ll, соответствующие им тройки задаются уравнениями последовательного преобразования [см. рис. 9.7 (b)].

Шаг 4. Если в графе появились параллельные связи, исключить их с помощью уравнений параллельного преобразования из табл. 9.10. В нашем случае параллельных связей создано не было.

Шаг 5. Если появились циклы, исключить их с помощью уравнений преобразования циклов из табл. 9.10. Удаление вершины j привело к созданию изолированного цикла ll, который исключается модификацией метки дуги ln, как требуют уравнения преобразования цикла. В этом месте дуги il и ln последовательно связаны и могут быть объединены, что ведет к исключению вершины l [см. рис. 9.7 (c)]. Когда дальнейшие исключения невозможны, возвратиться к шагу 1.

Эти пять шагов следует повторять до тех пор, пока граф не сведется только к вершинам входа и выхода. Метка дуги, соединяющей вход с выходом, содержит вероятность выхода через эту вершину выхода при условии входа в программу в данной точке входа. Там также содержится среднее время выполнения между этими двумя вершинами и его дисперсия. Применение описанной выше процедуры иллюстрируется на рис. 9.7, причем на шаге 1 последовательно выбираются для удаления вершины j, q, i, g. Результаты на рис. 9.7 (e) показывают, что среднее время выполнения равно 93 932 единицам времени и среднее квадратичное отклонение имеет такой же порядок, несмотря на предположение, что для всех дуг дисперсия равна нулю. Очевидно, что чувствительность этих чисел к точности оценок вероятностей велика.

Читателям предлагается убедиться в этом, изменяя три различные вероятности (p_{hi}, p_{ij} и $p_{ni}=p_{rq}$) при применении описанной процедуры к модифицированным моделям (см. задачу 9.10).

■

Модель, аналогичную описанной в примере 9.4, имитировать не трудно. Однако ее имитация не имеет, по-видимому, существенных преимуществ перед аналитическим решением, за исключением, возможно, того, что можно быстрее получить информацию, где тратится бóльшая часть времени выполнения программы. С другой стороны, имитация, безусловно, дороже в смысле вычислительного времени, чем аналитические методы, которые мы описали, и, по-видимому, удобна лишь для более сложных моделей (например, немарковских).

В области детерминированных моделей программ имитационное моделирование, надо полагать, безусловно, превосходит аналитические методы, или, более точно, не имеет альтернативы. Решения в детерминированной модели не базируются на заданных вероятностях ветвления, а зависят только от значений фактических предикатов. Поэтому модель этого типа должна явно включать все предикаты. Знания значений предикатов в каждый момент выполнения программы достаточно для определения пути передачи управления. Назовем *переменными управления* те переменные программы, которые появляются по крайней мере в одном предикате, и *состоянием управления* множество возможных значений переменных управления. Чтобы быть полной, модель должна специфицировать, как каждый блок последовательных операторов влияет на состояние управления программы. В программе из табл. 9.1 имеются (после подходящей замены аргументов подпрограммы) следующие предикаты: (IENT=0), (I⩽IENT), (J⩽JNT) и (II⩽IENT). Состояние управления программы в некоторый момент задается значением вектора (IENT, JNT, I, J, II) в этот момент. Блок, состоящий из операторов 1 и 2, присваивает переменным IENT и JNT значения, считываемые с вводимой карты, но не изменяет другие переменные управления. Блок, составленный из операторов от 21 до 26, никак не влияет на состояние управления программы. Заметим, что возможно косвенное влияние, которое также должно быть представлено в модели. Если бы существовал оператор 6′ вида IENT=IENT+INT(H), то все блоки, включающие операторы, которые устанавливают или модифицируют значение H (в нашем примере только блок оператора 1), и их влияние на H должны были бы специфицироваться в модели, хотя H и не является переменной управления.

Короче говоря, детерминированную модель программы можно построить в виде графовой модели, такой, например, как на рис. 9.6(b). Для определения времени выполнения и трассировочной записи для программы (а следовательно, также профиля и других индексов производительности) необходимо и достаточно специфицировать время выполнения для каждой вершины, ее прямое или косвенное влияние на состояние управления и предикат, который может быть на ее выходе.

Теперь мы покажем, что если нас интересует профиль программы, то имеется возможность написать уравнения для счетчиков частоты различных блоков и при сравнительно простых структурах программ решить их аналитически. Решения выражают счетчики частоты как функции от исходных данных и позволяют определить профиль программы и ряд важных индексов производительности программ (включая и время выполнения) для любого заданного множества значений исходных данных. В тех случаях, когда решение *уравнений профиля* очень трудно или невозможно, следует искать приближенное решение или прибегнуть к моделированию. В большой программе аналитические формулы этого типа можно выводить для частей программы и использовать для ускорения процесса имитации программы (см. задачу 9.14).

Недостатки такого аналитического подхода и приближенные методы их компенсации полностью еще не исследованы. Мы опишем этот метод в применении к простой программе из табл. 9.1. Но прежде нам нужно преобразовать программу, так чтобы ее блок-схема стала *Д-картой*. Д-карта [названная так в честь Дейкстры (1972), который впервые ввел ее] это блок-схема, правила формирования которой ограничиваются тремя конструкциями из рис. 9.8: *композиция*, *выбор* и *итерация*. Более точно, Д-карта — это ориентированный граф, вершины которого принадлежат одному из пяти типов: прямоугольники, представляющие блоки последовательных операторов или другие более сложные Д-карты (например, прямоугольники Z на рис. 9.8); ромбовидные вершины, представляющие выбор; круглые вершины, которые представляют соединения, и два вида треугольных вершин, представляющие точки входа и выхода. Если X и Y — Д-карты, то все три конструкции на рис. 9.8, каждая из которых может быть представлена одним прямоугольником Z, являются Д-картами. Эти конструкции мы будем называть *элементарными Д-картами*. Конструкции языка, соответствующие элементарным Д-картам, это три кита в программировании без оператора GOTO: нормальная последовательность операторов (X; Y), условный оператор **if** q_Z **then** X **else** Y и цикл **while** q_Z **do** X. Отметим, что на рис. 9.8 и далее предикат — это булева переменная, значение которой равно 1, если предикат истинен, и 0, если он ложен. Заметим также, что все элементарные Д-карты на рис. 9.8 имеют только один вход и один выход.

Когда графовая модель программы построена в форме Д-карты, уравнения профиля можно написать, заменяя итеративно каждую элементарную Д-карту прямоугольником с одним входом и одним выходом и применяя соответствующие уравнения на рис. 9.8. Теперь объясним символы, используемые в этих уравнениях. Пусть X — прямоугольник, а Z — другой прямоугольник, который заменяет элементарную Д-карту, содержащую X в упомянутой выше итеративной процедуре. Обозначим через nx число выполнений X при каждом входе в Z, а через NX общее число выполнений X в течение выполнения программы. Другими словами, NX — это счетчик ча-

стоты X. Так как число входов в Z равно NZ и каждый раз значения nx могут быть различными, обозначим эти значения через $nx(jz)$, где $jz=1, 2, \ldots, NZ$. Очевидно, что

$$NX=\sum_{jz=1}^{NZ} nx(jz). \tag{9.7}$$

Заметим, что jz, счетчик числа вхождений в прямоугольник Z, увеличивается при передаче управления на Z и вначале равен нулю. Назовем *jz индексом выполнения Z*.

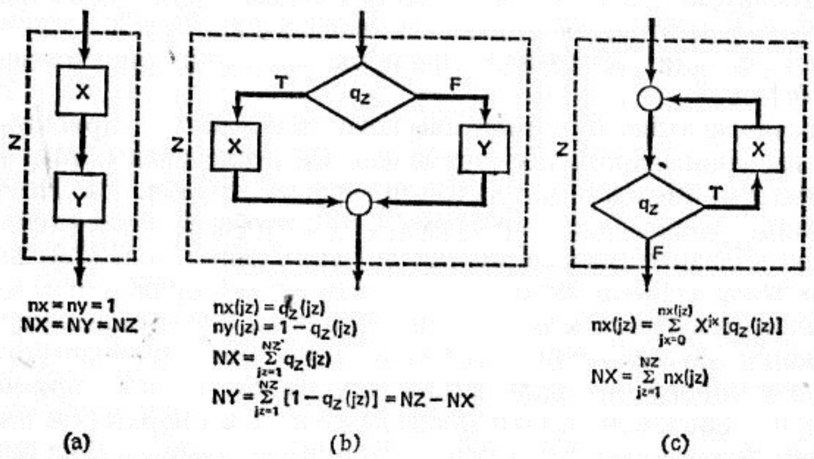


Рис. 9.8. Элементарные Д-карты и уравнения их профиля: (a) — композиция; (b) — выбор; (c) — итерация.

Уравнение для $nx(jz)$ на рис. 9.8(c) требует некоторых дополнительных объяснений. Через $q_Z(jz)$ мы обозначаем значение предиката q_Z, когда в прямоугольник Z производится вхождение jz-й раз. $X(q_Z(jz))$ есть значение $q_Z(jz)$ после первого выполнения прямоугольника X. Значение $q_Z(jz)$ после jx-го выполнения X обозначается через $X^{jx}[q_Z(jz)]$, и мы устанавливаем $X^0[q_Z(jz)]=q_Z(jz)$. Таким образом, уравнение для $nx(jz)$ на рис. 9.8(c) просто выражает тот факт, что если мы применяем X $nx(jz)$ раз к $q_Z(jz)$, то значение предиката будет ложным первый раз с момента входа в Z. Иначе говоря, при $nx(jz)$-м применении функции X значение $q_Z(jz)$ становится равным нулю. Аналитическая форма выражения для $nx(jz)$ может быть получена только тогда, когда q_Z и X специфицируются. Например, в простом цикле DO, где индекс I изменяется от начального значения INIT до граничного LIM с шагом 1, если $q_Z=$(I<LIM), то мы имеем

$$nx(jz)=(\mathrm{INIT}<\mathrm{LIM})+(\mathrm{INIT}+1<\mathrm{LIM})+\ldots \\ \ldots+(\mathrm{INIT}+nx(jz)<\mathrm{LIM}), \tag{9.8}$$

где последний предикат и только он должен иметь значение ложь. Далее, если INIT<LIM, то

$$\mathrm{INIT}+nx(jz)=\mathrm{LIM} \tag{9.9}$$

и отсюда

$$nx(jz)=\mathrm{LIM}-\mathrm{INIT}. \tag{9.10}$$

Если, с другой стороны, INIT≥LIM, то

$$nx(jz)=0. \tag{9.11}$$

Уравнения (9.10) и (9.11) могут быть записаны в более компактной форме

$$nx(jz)=(\mathrm{LIM}-\mathrm{INIT})(\mathrm{LIM}\geqslant\mathrm{INIT}), \tag{9.12}$$

где (LIM≥INIT) равно 1, если заключенное в скобки неравенство верно, и равно 0, если оно ложно. Аналогично, если предикат q_Z есть (I≤LIM), то уравнение (9.12) имеет вид

$$nx(jz)=(\mathrm{LIM}-\mathrm{INIT}+1)(\mathrm{LIM}\geqslant\mathrm{INIT}). \tag{9.13}$$

Заметим, что число выполнений цикла DO равно $nx(jz)+1$, где $nx(jz)$ задается формулами (9.10) или (9.11). Это несоответствие связано с различием структуры итерации в ФОРТРАНе и структуры элементарной Д-карты на рис. 9.8(c). Заметим также, что $nx(jz)$ в (9.10) зависит от jz, если LIM или INIT зависят от jz.

Прямоугольники Z на рис. 9.8 могут в свою очередь являться частью большей Д-карты, которая в итеративной процедуре замещения, иллюстрируемой в примере 9.5, должна быть заменена бо́льшим прямоугольником, скажем W. Каждый раз при входе в W блок Z выполняется $nz(jw)$ раз, где $nz(jw)$ аналогично $nx(jz)$ из той же самой элементарной Д-карты на рис. 9.8. Для каждого выполнения W индекс jz пробегает значения от 1 до $nz(jw)$ и устанавливается в 0 каждый раз при выходе из W. Таким образом, X выполняется столько раз, каково значение

$$nx(jw)=\sum_{jz=1}^{nz(jw)} nx(jz,\ jw), \tag{9.14}$$

и общее число выполнений X (пока W в свою очередь не заменен бо́льшим прямоугольником) равно

$$NX=\sum_{jw=1}^{NW}\ \sum_{jz=1}^{nz(jw)} nx(jz,\ jw). \tag{9.15}$$

Если W заключен в некий прямоугольник V, то NX и NW в формуле (9.15) должны быть переписаны в виде $nx(jv)$ и $nw(jv)$. Кроме того, $nx(jz,\ jw)$ и $jz(jw)$ нужно заменить соответственно на $nx(jz,\ jw,\ jv)$ и $nz(jw,\ jv)$.

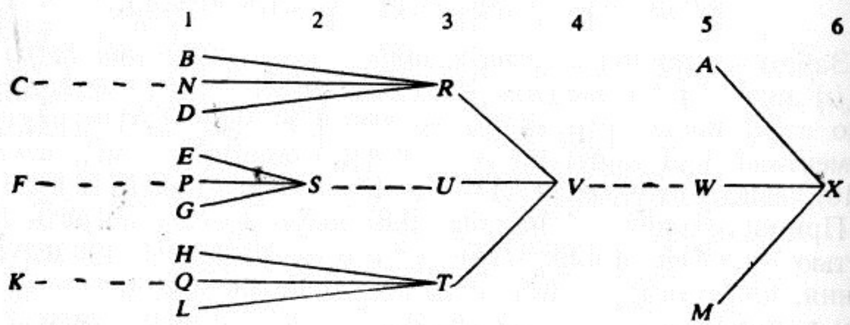
Пример 9.5. Программа в табл. 9.1 записана в форме, мало удобной для представления в виде Д-карты, но легко преобразуема в нужную форму. На рис. 9.9 (a) приведена Д-карта программы, эквивалентной первоначальной. Читателю не доставит труда понять, как было сделано это преобразование и как оно скажется на тексте программы. Рядом с каждым прямоугольником в круглых скобках указаны номера операторов, которые он представляет. Заметим, что структурные несоответствия меж-

ду языком ФОРТРАН и языком, ориентированным на Д-карты, так же как и различия итераций, о которых мы говорили выше, требуют, чтобы некоторые операторы появлялись более чем в одном прямоугольнике. Впрочем, в этом нет особенных трудностей, за исключением небольшого увеличения сложности Д-карты. Конечно, мы должны помнить это при вычислении числа выполнений определенных операторов, скажем оператора 21. Так как оператор 21 выполняется 1 раз при каждом входе в B, C, E или F, то $N21 = NB + NC + NE + NF$. На табл. 9.11 (d) приведена взаимосвязь между счетчиками частоты в Д-карте на рис. 9.9 (a) и счетчиками частоты в графовой

Таблица 9.11

Соответствие между Д-картой на рис. 9.9 (a) и блок-схемой и графовой моделью на рис. 9.6

(a) *Определения прямоугольников более высокого уровня на различных шагах процедуры итеративного замещения* (непрерывная линия — композиция, пунктирная линия — итерация)



(b) *Предикаты*

$q_N = (J < JNT)$	
$q_P = (J < JNT)$	
$q_U = (II \leqslant IENT)$	
$q_Q = (I < IENT)$	
$q_W = (IENT \neq 0)$	

(c)

Переменные управления	*Устанавливаются в*	*Получают приращения в*
IENT	A, L	—
JNT	A, L	—
J	B, E	C, F
II	B	D, G
I	H	K

(d) *Связь счетчиков частоты*

$$
\begin{aligned}
NI &= NA + NL \\
NII &= NB \\
NIII &= NB + NE \\
NIV &= NB + NC + NE + NF \\
NV &= ND + NG \\
NVI &= NH \\
NVII &= NH + NK
\end{aligned}
$$

модели на рис. 9.6 (b), где каждый оператор принадлежит только одной вершине. В кружочке внутри каждого прямоугольника на рис. 9.9 (a) мы приводим время выполнения этого прямоугольника, соответствующее времени, приведенному в табл. 9.2. Заметим, что предполагается, что вершины выбора и соединения выполняются за время, равное нулю.

В табл. 9.11 (b) и (c) содержится вся информация, которая должна прилагаться к Д-карте, чтобы наша задача стала разрешимой. В части (b) таблицы специфицируются пять предикатов, имеющихся на рис. 9.9 (a), в части (c) таблицы приводятся характеристики прямоугольников Д-карты в терминах их влияния на состояние управления программы. Причина, почему имена предикатов (q_N, q_P, . . .) выбраны именно

так, станет ясна, если читатель проследит за работой процедуры итеративного замещения по табл. 9.11 (a). На каждом шаге производятся все возможные замены элементарных Д-карт прямоугольниками более высокого уровня. Например, в Д-карте на рис. 9.9 (a) имеется три итерационных цикла, аналогичных циклу на рис. 9.8 (c).

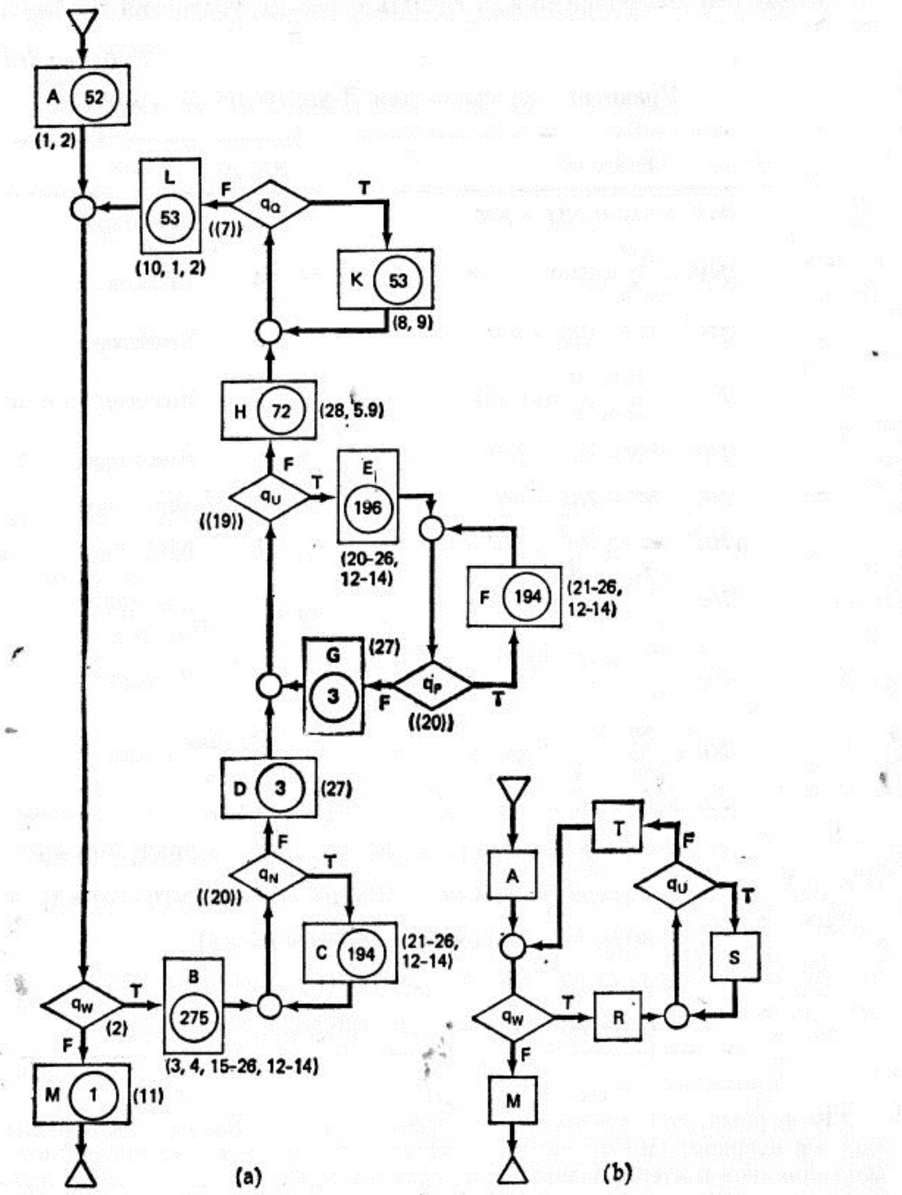


Рис. 9.9. Модели с использованием Д-карт для программы из табл. 9.1: (a) — на шаге 0; (b) — на шаге 3 процедуры итеративного замещения [см. табл. 9.11 (a)].

Последний с прямоугольником C и выбором q_N будет заменен прямоугольником N, и аналогично F и q_P заменяется на P, а K и q_Q — на Q. Далее мы получим прямоугольники R, S и T, применяя композицию, и т. д., пока вся Д-карта не сведется к одному единственному прямоугольнику X. На рис. 9.9 (b) изображена Д-карта как раз после введения R, S и T.

Теперь, когда построена иерархия Д-карт методом снизу вверх, можно написать уравнения профиля. Для этого удобно действовать в направлении сверху вниз, начиная с X и продвигаясь к листьям дерева, как показано в табл. 9.12. Уравнения в этой таблице получены из уравнений на рис. 9.8 и уравнений (9.12) — (9.15).

Несколько замечаний нужно сделать о выводе уравнений профиля.

Таблица 9.12

Уравнения профиля для Д-карты на рис. 9.9(а)

Уравнение	Шаг	Тип
$NA = NW = NM = NX$	5	Композиция
$NV = \sum_{jw=1}^{NW} nv(jw)$	4	Итерация
$NR = NU = NT = NV$	3	Композиция
$NS = \sum_{jw=1}^{NW} \sum_{ju=1}^{nu(jw)} ns(jw, ju)$	2	Итерация
$NB = NN = ND = NR$	1	Композиция
$NE = NP = NG = NS$	1	Композиция
$NH = NQ = NL = NT$	1	Композиция
$NC = \sum_{jw=1}^{NW} \sum_{jn=1}^{nn(jw)} nc(jw, jn)$	0	Итерация
$NF = \sum_{jw=1}^{NW} \sum_{ju=1}^{nu(jw)} \sum_{jp=1}^{np(jw, ju)} nf(jw, ju, jp)$	0	Итерация
$NK = \sum_{jw=1}^{NW} \sum_{jq=1}^{nq(jw)} nk(jw, jq)$	0	Итерация

$nv(jw) = nu(jw) = nn(jw) = nq(jw) = \#(\mathrm{IENT}_i \neq 0) = \nu$

$ns(jw, ju) = np(jw, ju) = [\mathrm{IENT}(jw, ju) - 1][\mathrm{IENT}(jw, ju) \geq 1]$

$nc(jw, jn) = [\mathrm{JNT}(jw, jn) - 1][\mathrm{JNT}(jw, jn) \geq 1]$

$nf(jw, ju, jp) = [\mathrm{JNT}(jw, ju) - 1][\mathrm{JNT}(jw, ju) \geq 1]$

$nk(jw, jq) = [\mathrm{IENT}(jw, jq) - 1][\mathrm{IENT}(jw, jq) \geq 1]$

Во-первых, эти уравнения содержат столько строчных переменных, сколько имеется итераций, и столько индексов выполнения, сколько предикатов. Пять прямоугольников в итерировании Д-карт это V, S, C, F и K, а соответствующие строчные переменные nv, ns, nc, nf и nk. Имена пяти предикатов q_W, q_U, q_N, q_P и q_Q, и соответствующие индексы выполнения jw, ju, jn, jp и jq.

Во-вторых, у прямоугольников с взаимоотношением композиции не только одинаковые конечные счетчики частоты, но также одинаковые текущие счетчики частоты, при условии что они пересчитываются при выходе из наименьшего заключающего их прямоугольника. Например, существует отношение композиции между V, U, N и Q [все они связаны непрерывными линиями на дереве в табл. 9.11 (а)]. Поэтому у нас не только $NV=NU=NN=NQ$, но и $nv=nu=nn=nq$. Так как nv — это число выполнений V при входе в W, следует ожидать, что jw есть функция индекса выпол-

нения W, обозначаемого jw, и, следовательно, мы пишем $nv(jw)=nu(jw)=nn(jw)=nq(jw)$.

В-третьих, W — это итерация, но не цикл DO; qw проверяет значение IENT, устанавливаемое оператором A в первый раз и затем оператором L каждый раз при выполнении V. Поэтому удобно обозначить последовательность IENT_1, IENT_2, ...

Таблица 9.13

Решения уравнений профиля из табл. 9.12 для $NX=1$, $\mathrm{IENT}_i \geq 1$, $\mathrm{JNT}_i \geq 1$ $(i=1,2, \ldots, v)$

$NA = NW = NM = 1$

$NV = nv(1) = \nu$

$NR = NU = NT = \nu$

$NS = \sum_{ju=1}^{\nu} \mathrm{IENT}(1, ju) - \nu$

$NB = NN = ND = \nu$

$NE = NP = NG = \sum_{ju=1}^{\nu} \mathrm{IENT}(1, ju) - \nu$

$NH = NQ = NL = \nu$

$NC = \sum_{jn=1}^{\nu} \mathrm{JNT}(1, jn) - \nu$

$NF = \sum_{ju=1}^{\nu} \sum_{jp=1}^{\mathrm{IENT}(1,ju)-1} [\mathrm{JNT}(1, ju) - 1] = \sum_{ju=1}^{\nu} [\mathrm{IENT}(1, ju) - 1][\mathrm{JNT}(1, ju) - 1]$

$NK = \sum_{jq=1}^{\nu} \mathrm{IENT}(1, jq) - \nu$

$\mathrm{NI} = 1 + \nu$

$\mathrm{NII} = \nu$

$\mathrm{NIII} = \sum_{ju=1}^{\nu} \mathrm{IENT}(1, ju)$

$\mathrm{NIV} = \sum_{ju=1}^{\nu} \mathrm{IENT}(1, ju) \cdot \mathrm{JNT}(1, ju)$

$\mathrm{NV} = \sum_{ju=1}^{\nu} \mathrm{IENT}(1, ju)$

$\mathrm{NVI} = \nu$

$\mathrm{NVII} = \sum_{jq=1}^{\nu} \mathrm{IENT}(1, jq)$

$t = NA \cdot TA + NB \cdot TB + \cdots + NM \cdot TM$

$= 194 \sum_{ju=1}^{\nu} \mathrm{IENT}(1, ju) \cdot \mathrm{JNT}(1, ju) + 58 \sum_{ju=1}^{\nu} \mathrm{IENT}(1, ju) + 151\nu + 53$

значений IENT двумерной величиной, например IENT (jw, jv). Так как значение jv получает приращение при входе в V и устанавливается равным 0 каждый раз при выходе из W, IENT (1,1) внутри V будет обозначать первое значение IENT (т. е. IENT_1), IENT(1,2) будет соответствовать второму (IENT_2), IENT (1,nv(1)) — $\mathrm{IENT}_{nv(2)}$,

IENT(2,1)—IENT$_{nv(1)+1}$ и т. д. Аналогично, внутри U текущее значение IENT, значение только что проверенное предикатом q_w, можно обозначить через IENT(jw, ju), а внутри Q — через IENT(jw, jq). Аналогичная система обозначений может быть применима и к JNT.

В-четвертых, число последовательных вводимых карт, в которых IENT$\neq$0, является начальным данным нашей задачи и было обозначено через ν в табл. 9.12.

Решение уравнений профиля для этой простой программы не сложно. Решение для NX=1, IENT$_i\geqslant$1 и JNT$_i\geqslant$1 (i=1, . . . , ν) приведено в табл. 9.13. В этой же таблице приводятся выражения NI ,..., NVII, полученные из выражений NA ,..., NK в этой же таблице и из уравнений в табл. 9.11 (d). Эти формулы позволяют нам вычислить те же счетчики частоты, которые мы могли бы измерить с помощью инструментации в табл. 9.7 (d) и по которым мы можем построить профиль программы, в соответствии с определением в разд. 9.2.1. Зная время выполнения TJ каждого прямоугольника J в полной Д-карте [см. рис. 9.9 (a)], мы можем очень быстро вычислить время, затрачиваемое в J, для любого множества значений входных данных. Сумма всех этих времен есть время выполнения программы, для которого в табл. 9.13 мы даем формулу вычисления по значениям времен операторов из таблицы 9.2. Пользуясь этой формулой, читатель может удостовериться, что время выполнения для IENT$_1$=1, JNT$_1$=2, IENT$_2$=0 есть время, полученное по трассировочной записи в табл. 9.2 (t=650 единицам времени). Также для IENT$_i$=10, JNT$_i$=5 (i=1, . . ., 9)

Таблица 9.14

Модификации к табл. 9.12 и 9.13, когда оператор 20 заменяется на оператор DO 1 J = II,K

Для табл. 9.12:

$$nf(jw, ju, jp) = [\text{JNT}(jw, ju) - jp - 1][\text{JNT}(jw, ju) \geq (jp + 1)]$$

Для табл. 9.13:

$$NF = \sum_{ju=1}^{\nu} \sum_{jp=1}^{\text{IENT}(1,ju)-1} [\text{JNT}(1, ju) - jp - 1][\text{JNT}(1, ju) \geq (jp + 1)]$$

$$= \sum_{ju=1}^{\nu} (\{\text{IENT}(1, ju)[\text{JNT}(1, ju) - \tfrac{1}{2}\text{IENT}(1, ju) - \tfrac{1}{2}] - \text{JNT}(1, ju) + 1\}$$

$$\cdot [\text{JNT}(1, ju) \geq \text{IENT}(1, ju)]$$

$$+ \{\tfrac{1}{2}[\text{JNT}(1, ju) - 2][\text{JNT}(1, ju) - 1]\}[\text{JNT}(1, ju) < \text{IENT}(1, ju)])$$

$$\text{NIV} = \sum_{ju=1}^{\nu} \text{IENT}(1, ju)[\text{JNT}(1, ju) - \tfrac{1}{2}\text{IENT}(1, ju) + \tfrac{1}{2}] \quad \text{if} \quad \text{JNT}(1, ju) \geq \text{IENT}(1, ju)$$

$$= \sum_{ju=1}^{\nu} \{\text{IENT}(1, ju) + \tfrac{1}{2}\text{JNT}(1, ju)[\text{JNT}(1, ju) - 1]\} \quad \text{if} \quad \text{JNT}(1, ju) < \text{IENT}(1, ju)$$

$$t = 194 \sum_{ju=1}^{\nu} \text{IENT}(1, ju) \cdot \text{JNT}(1, ju) - 97 \sum_{ju=1}^{\nu} [\text{IENT}(1, ju)]^2 + 155 \sum_{ju=1}^{\nu} \text{IENT}(1, ju)$$

$$+ 151\nu + 53 \quad \text{if JNT}(1, ju) \geq \text{IENT}(1, ju) \quad \text{for all } ju$$

$$= 252 \sum_{ju=1}^{\nu} \text{IENT}(1, ju) + 97 \sum_{ju=1}^{\nu} [\text{JNT}(1, ju)]^2 - 97 \sum_{ju=1}^{\nu} \text{JNT}(1, ju) + 151\nu + 53$$

$$\text{if JNT}(1, ju) < \text{IENT}(1, ju) \quad \text{for all } ju$$

формула дает время выполнения, равное 93 932 единицам времени, что совпадает со средним временем выполнения, найденным в примере 9.4 [см. рис. 9.7 (e)].

Применение этого метода к несколько более сложной программе иллюстрируется в таблице 9.14. Новая программа идентична той, которую мы только что проанализировали, за исключением оператора 20, который теперь имеет вид DO 1 J=II, K. Начальное значение индекса J в цикле DO устанавливается в B равным 1 и увеличивается в D и G. Поэтому для прямоугольника C ничего не меняется, но по отношению к P мы имеем

$$\text{II}(jp) = jp + 1, \tag{9.16}$$

так как II равно 2, когда передача управления приходит на U (и, следовательно, также на P), и увеличивается на 1 в G. Уравнение для nf в табл. 9.14 выводится из (9.12). Формулы для NF, NIV и t можно вывести с помощью некоторых преобразований из уравнения для nf и из предыдущих формул в табл. 9.13. Эти решения были получены при тех же предположениях, что и для табл. 9.13.

Формулы, полученные этим методом, безусловно, представляют интерес в различных контекстах. Очевидно, однако, что для больших программ этот анализ следует автоматизировать или по крайней мере существенно полуавтоматизировать и что предпочтительнее подходящие приблизительные методы для тех случаев, когда точные решения получить очень трудно. ■

Обсуждение индексов поведения программы в разд. 9.2.2 показало, что существует ряд различных характеристик, относящихся к поведению программы. Эти характеристики охватывают широкий спектр степеней детализации и зависимостей от стратегий управления памятью. На одной границе этого спектра строка ссылок, являющаяся очень подробной и не зависящей от стратегии моделью поведения программы. На другой — один скалярный индекс, такой, как, например, число страничных отказов, значения которого характеризуют фактически не пару (программа, входные данные), а скорее комбинацию программы, ее входных данных и конкретной стратегии, обслуживающей прохождение программы. Понятие локальности от стратегии не зависит, но также очень трудно измеряемо не зависящим от стратегии способом.

Единственный метод анализа производительности, в котором может использоваться строка ссылок программы, это имитационная модель, управляемая трассировочной записью. Для других типов имитационного моделирования и аналитических методов требуется более точная модель поведения программы. По-видимому, невозможно сохранить всю существенную информацию, которую содержит строка, при значительном сокращении ее размера, если стратегию никак не специфицировать. Поэтому работа по разработке моделей поведения программы должна развиваться в направлениях специализации и аппроксимации.

Специализированные модели — это модели, которые конструируются для заданного класса стратегий. Они могут быть точными, в том смысле, что в них присутствует вся существенная информация. Другими словами, они могут давать точные результаты при вычислении определенных индексов производительности программ. Но

нельзя или неудобно применять их к другим задачам или к стратегиям не из этого класса.

Простым примером такой модели является *кривая жизни*, введенная в примере 7.3. Ее использование ограничивается вычислением общего числа страничных отказов, генерируемых программой, или

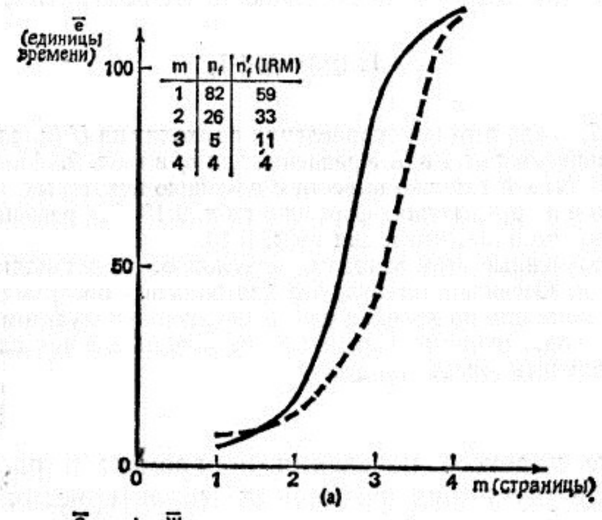


Рис. 9.10. Простые модели поведения программы для строки ссылок на страницы из табл. 9.15 (a): (a) — кривые жизни LRU, полученные по строке (сплошная линия) и по прогнозу IRM (пунктирная линия); (b) — кривая жизни в области времени ($\bar{e}$) и характеристика размера рабочего множества ($\bar{w}$), полученные по строке.

ее средней интенсивности отказов за все прохождение задачи, когда схема управления памятью принадлежит классу стратегий фиксированного распределения со страничной обработкой по требованию и специфическим алгоритмам замещения (например, LRU). Любое применение кривой жизни вне этой области либо невозможно, либо является грубым приближением. Примерами последнего является

наше применение кривой жизни в нескольких задачах в гл. 7 и 8. Как отмечено в разд. 9.3.1, алгоритм из табл. 9.8 можно слегка изменить для получения для программы кривой жизни LRU за один просмотр строки ссылок программы.

Пусть, например, задана строка ссылок на блоки в табл. 9.4(a) и отображение $\pi_1=(b_1, b_3)$, $\pi_2=(b_5)$, $\pi_3=(b_7, b_8)$, $\pi_4=(b_2, b_4, b_6)$ восьми блоков программы на четыре страницы π_1, π_2, π_3, π_4, по которым мы получаем строку ссылок на страницы в табл. 9.15(a). Отсюда мы легко выводим график кривой жизни на рис. 9.10(a). Он позволяет нам вычислить число страничных отказов, генерируемых парой (программа, исходные данные) для заданного ресурса распределения памяти m, но только тогда, когда этот ресурс остается постоянным в течение всего выполнения и когда управление памятью производится в соответствии с алгоритмом LRU. Отметим, что применяя этот алгоритм к строке ссылок на страницы в табл. 9.15(a), мы предполагали, что если программа генерирует другие ссылки, которые в строке отсутствуют, то все они относятся к самой последней из использованных страниц. Поэтому строка 1 3 4 могла бы на самом деле быть строкой 1 1 1 1 1 3 3 3 4 4 4 4, и наши результаты остались бы правильными. Это предположение не очень реалистично. В строке, генерируемой на уровне машинного языка, выбор команд часто чередуется с обращениями к операндам. Поэтому приведенная выше строка более вероятно имела бы вид 1 4 1 1 4 3 4 3 4 4 3 4. Если чередование выбора ограничено двумя страницами, как в нашей строке, то только значение $\bar{e}$ для $m=1$ на рис. 9.10(a) не точное и больше действительного, так как при обработке строки из табл. 9.15(a) число страничных отказов, генерируемых для $m=1$, недооценивается. Заметим также, что было предположено, что отведенная программе память вначале пуста.

Таблица 9.15

Строка ссылок на страницы, полученная по строке ссылок на блоки из табл. 9.4.

(a) Строка ссылок на страницы (времена ссылок, как в табл. 9.4)

1341414314242424242424141412

4241414124241414124241414124 24

2424214141414143113414 1

(b) Строка множеств ссылок на страницы (одно множество на каждые 25 единиц времени)

(1, 3, 4)(3, 4)(1, 3, 4)(1, 3, 4)(1, 2, 4)(1, 2, 4)(1, 2, 4)(1, 2, 4)(1, 2, 4)(1, 4)(1, 2, 4)

(1, 2, 4)(2, 4)(1, 2, 4)(1, 3, 4)(1, 3, 4)(1, 3, 4)(1, 3, 4)

Другими моделями того же типа, которые корректны для стратегии рабочего множества с постоянным размером окна τ, являются кривая $\bar{e}(\tau)$, которую можно назвать *кривой жизни в области времени*, *и характеристика размера рабочего множества* $\bar{w}(\tau)$, где $\bar{w}$ — средний размер рабочего множества программы. Как и $\bar{e}(m)$, эти кривые можно измерить и построить с помощью имитационного моделирования, управляемого трассировочной записью. На рис. 9.10(b) приведены обе кривые, полученные по строке множеств ссылок на страницы из табл. 9.15(b), которая соответствует строке ссылок на блоки из табл. 9.4(b) и описанному выше отображению. Заметим, что, когда строка состоит из множеств ссылок, а не из ссылок, алгоритм в табл. 9.9 должен быть модифицирован с учетом того, что окно может продвигаться каждый раз только на одно множество ссылок, а не на одну ссылку. Поэтому окно состоит из целого числа множеств, и на каждом шаге может генерироваться более чем один страничный отказ. Если мы используем строку множеств ссылок, мы получаем кривые, являющиеся лишь приближением кривых, которые получились бы при обработке простой стратегией рабочего множества из табл. 9.9 полной строки ссылок. Впрочем, наиболее практичные реализации стратегий рабочего множества измеряют рабочее множество не каждый раз при генерации ссылки, а лишь периодически. Если этот период и размер окна кратны интервалу, соответствующему множеству ссылок, то использование строки множеств ссылок не влияет на подсчет числа страничных отказов и дает более реалистичную оценку объема памяти, которая в действительности потребуется программе. Кроме того, здесь имитация, как правило, проходит быстрее, чем в случае использования полной строки ссылок.

Если желательна модель более компактная, чем строка ссылок, но не ограниченная только неким классом стратегий, то необходимо делать упрощающие предположения, и результирующая модель поэтому будет представлять поведение программы только приблизительно. В принципе это не является проблемой, так как любая модель любой системы есть всегда ее приблизительное представление. Проблема заключается в адекватности модели, о чем можно судить лишь по отношению к некоторому критерию, в зависимости от точки зрения экспериментатора и предполагаемого применения модели.

При оценке производительности программы главным критерием должна быть способность модели предсказывать время выполнения данной программы. Это означает, что число страничных отказов, которые будет генерировать программа с любой заданной стратегией управления памятью, должно быть предсказуемо. Впрочем, иногда имеются и другие заслуживающие внимания критерии. Экспериментатор может, например, быть более заинтересован в доказательстве общих утверждений о стратегиях, чем в вычислении числа страничных отказов, получаемых конкретной программой под управлением конкретной стратегии. В этом случае простая аналитическая модель,

динамическое поведение которой в какой-то степени сходно с поведением некоторых конкретных программ, безусловно, более подходяща, чем более подробная, но математически не прослеживаемая модель. Результаты основанных на методах из разд. 9.3.1 экспериментов по измерению программ должны использоваться не только для калибровки и контроля более подробных моделей, но и служить источником вдохновения и руководством при формулировке даже самых грубых и самых общих моделей.

Распространенным способом получения точного представления поведения сложной системы во времени является построение ее стохастической модели. Это естественный подход и к моделированию строк ссылок для программ. В общем стохастическое моделирование включает определение состояний моделируемой системы и их вероятностей или вероятностей возможных переходов между ними. Ключевым моментом здесь является выбор состояний, он должен делаться с учетом целей моделирования и индивидуальных характеристик системы. В простейшем случае состояния программы в любой заданный момент идентифицируются самой последней страницей (или блоком), на которую делалась ссылка. В этом случае имеется столько состояний, сколько в программе страниц. Если предполагается, что появление каждого состояния статистически не зависит от появления всех других состояний и от времени, то мы имеем дело с *моделью независимых ссылок* (IRM). Следовательно, IRM характеризует поведение программы, у которой u страниц, распределением вероятностей ссылок на структуру $\{p_i\}$, которое подчиняется соотношению

$$\sum_{i=1}^{u} p_i = 1. \tag{9.17}$$

Другими словами, описание программы посредством IRM эквивалентно профилю страниц программы или блоков программы. Таблица 9.5, в которой приведены профили блоков строки из табл. 9.4(a) и двух ее подстрок, показывает, почему IRM не является адекватной моделью поведения программы. Модель IRM не способна представить фундаментальное свойство локальности, так как она воспроизводит плотность использования страниц программы только в глобальном масштабе. Заметим, что IRM совпадает с профилем блоков только тогда, когда ссылки на блоки делаются через регулярные интервалы. Иначе нам следовало бы также для каждого состояния специфицировать вероятность выхода из него, что позволило бы модели представить переменные интервалы между последовательными ссылками.

Как можно использовать стохастическую модель типа IRM (и другие, которые мы опишем) для оценки производительности моделируемой программы в среде с виртуальной памятью? Для некоторых стратегий возможно получить формулы, которые выражают число страничных отказов или среднюю интенсивность страничных отказов как функцию от $p_1, \ldots, p_u$. Например, средняя интенсивность стра-

ничных отказов, генерируемых программой, представляемой IRM под управлением LRU, при m страницах физической памяти равна [см. Коффман и Деннинг (1973), стр. 272]

$$\bar{f} = \sum_{\text{все } \mathbf{s}} D_1^2(\mathbf{s}) \prod_{i=1}^{m} \frac{p_{ji}}{D_i(\mathbf{s})}, \tag{9.18}$$

где $\mathbf{s}=(j_1, j_2, \ldots, j_m)$ есть перестановка m индексов страниц, взятых из $(1, 2, \ldots, u)$ и

$$D_i(\mathbf{s}) = 1 - \sum_{k=1}^{m-i+1} p_{jk} \quad (i=1, \ldots, m). \tag{9.19}$$

Применение формулы (9.18) к строке ссылок на страницы из табл. 9.15(a) дает кривую жизни, изображенную пунктиром на рис. 9.10(a).

Если для стохастической модели аналитическую формулу получить не удается (с учетом стратегии), модель можно использовать для генерации строки ссылок в реальном времени для так называемого управляемого распределением имитатора иерархии памяти. Например, для IRM всегда, когда имитатором должна обрабатываться новая ссылка на страницу (см. шаг 2 алгоритмов из табл. 9.8 и 9.9), генерируется псевдослучайное число и с помощью метода, описанного в примере 3.3, по дискретному распределению $\{p_i\}$ получается ссылка.

Другой моделью, основанной на таком же выборе состояний программы, как и для IRM, является простая марковская модель, определяемая матрицей вероятностей переходов от страницы к странице или от блока к блоку. Оценку этих вероятностей можно сделать по строке ссылок. Если c_{jk} — счетчик частоты упорядоченной пары (π_i, π_k) в строке, то вероятность того, что после ссылки на страницу π_i программа будет ссылаться на страницу π_k, можно оценить по формуле

$$p_{ik} = \frac{c_{ik}}{\sum_{k=1}^{u} c_{ik}} \quad (i, k=1, \ldots, u). \tag{9.20}$$

Таким образом, динамическое поведение программы в этой модели характеризуется парой $(j_1, \{p_{ik}\})$, где j_1 — индекс первой используемой страницы. Матрицы счетчиков частоты c_{ik} и вероятностей переходов p_{ik}, полученные из строки ссылок на страницы в табл. 9.15(a), приводятся в табл. 9.16(a) и (b). Заметим, что элементы c_{ii} и p_{ii} были бы гораздо больше, если бы строка получалась на уровне машинного языка.

Выбор состояний программы, позволяющий достаточно хорошо моделировать свойство локальности, может основываться на позициях стека LRU, а не на индивидуальных страницах. Если мы пронумеруем позиции стека LRU от 1 до u, начиная с самой последней использованной страницы и определим *стековое расстояние LRU* от ссылки до страницы π_i как положение π_i в стеке LRU в момент образования ссылки, то можно отождествить состояние программы со значением этого расстояния. Заметим, что в алгоритме LRU,

Таблица 9.16

Матрицы счетчиков частоты и вероятностей переходов для двух марковских моделей строки ссылок на страницы из табл. 9.15(a)

Описание в терминах ссылок на страницы

От страницы \ На страницу	1	2	3	4
1	1	4	2	17
2	1	0	0	17
3	2	0	0	2
4	20	14	2	0

(a) c_{ik}

От страницы \ На страницу	1	2	3	4
1	0.042	0.166	0.084	0.708
2	0.055	0	0	0.945
3	0.500	0	0	0.500
4	0.555	0.389	0.056	0

(b) p_{ik}

Описание в терминах стекового расстояния LRU

От расстояния \ На расстояние	1	2	3	4
1	0	1	0	0
2	0	41	10	1
3	1	10	6	0
4	0	0	1	0

(c) c_{ik}

От расстояния \ На расстояние	1	2	3	4
1	0	1	0	0
2	0	0.788	0.192	0.020
3	0.059	0.588	0.353	0
4	0	0	1	0

(d) d_{ik}

описанном в табл. 9.8, стековое расстояние LRU есть переменная δ. Так как имеется $u+1$ возможных расстояний (первые u натуральных чисел и значение ∞, являющееся расстоянием, условно приписанным любой странице, на которую прежде не ссылались), то программа характеризуется нахождением в любой момент в одном из $u+1$ состояний.

Одной из очень распространенных моделей такого типа является так называемая *простая стековая модель LRU* (SLRUM). Модель SLRUM состоит из пар $(\mathbf{s}_0, \{d_i\})$, где $\mathbf{s}_0$ — начальное состояние стека LRU, а $\{d_i\}$ $(i=1, \ldots, u, \infty)$ — распределение вероятностей расстояний. Так как $\mathbf{s}_0$ задается, то предполагается, что на все страницы ссылались по крайней мере один раз перед началом моделируе-

мой строки. Поэтому $d_\infty=0$ и $\{d_i\}$ состоит из u вероятностей, сумма которых равна 1. Как и прежде, требуется задание дополнительной информации, например, понадобится распределение вероятностей длительности каждого состояния, если ссылки в строке, по которой оцениваются параметры модели, производились не через регулярные интервалы. Если это замечание не учитывалось, то модель строки ссылок на страницы из табл. 9.15 (a) может иметь следующий вид ((2, 4, 1, 3), (0.013, 0.818, 0.156, 0.013)). Заметим, что начало отсчета времени приурочено к моменту одиннадцатой ссылки, являющейся первой ссылкой на страницу π_2, так как стек LRU переполняется только в этой точке. Далее, значение d_1 очень мало из-за того, что так выбран тип рассматриваемой строки. В реальной строке, соответствующей уровню машинного языка, d_1 — часто самая большая вероятность расстояния, а d_i почти всегда резко уменьшаются с увеличением i.

Модель SLRUM может адекватно представить аспект локальности в поведении программы в связи с ее способностью концентрировать ссылки на малом числе страниц для больших промежутков времени, не приписывая фиксированные вероятности этим страницам. Допускается изменение элементов текущей локальной области, и любая новая страница с большой вероятностью будет оставаться в области в течение относительно большого периода. Другими словами, у каждой страницы вероятность ссылки на нее переменна. Это связано с тем фактом, что идентичность страниц в этой модели не учитывается. Модель SLRUM можно рассматривать как модель независимых расстояний, за исключением того, что в отличие от IRM требуется специфицировать начальное условие ($\mathbf{s}_0$), чтобы иметь возможность вывода строки ссылок из строки расстояний. Сам вывод прост и очевиден. Например, если $\mathbf{s}_j=(4, 2, 1, 3)$ и $\delta_{j+1}=3$, то $i_{j+1}=1$, иначе говоря, следующая страница в стеке, на которую делается ссылка, третья и $\mathbf{s}_{j+1}=(1, 4, 2, 3)$.

Модель SLRUM точно предсказывает среднюю интенсивность страничных отказов моделируемой программы, управляемой стратегией замещения LRU с фиксированным распределением памяти при m страницах физической памяти:

$$\overline{f}=1-\sum_{i=1}^{m} d_i. \tag{9.21}$$

Можно также доказать [см. Коффман и Деннинг (1973, стр. 276)], что для генерируемой моделью SLRUM строки, у которой

$$d_1 \geqslant d_2 \geqslant \ldots \geqslant d_n, \tag{9.22}$$

алгоритм LRU оптимален для всех $m \geqslant 1$ в том смысле, что никакой другой алгоритм замещения, не предполагающий предварительного

знания появления ссылок, не будет генерировать меньше страничных отказов, чем алгоритм LRU. Поэтому можно сказать, что стратегия LRU предполагает соответствие поведения программ модели SLRUM, или, иначе говоря, что модель SLRUM лежит в основе стратегии LRU.

Спирн и Деннинг (1972) обнаружили, что модель SLRUM может также предсказывать достаточно аккуратно характеристику размера рабочего множества программы. К сожалению, резкая смена локальной области, характерная для поведения большинства программ, в модели SLRUM представлена не достаточно хорошо. Так как эта модель может давать строки только с очень плавной и медленной сменой, ее нельзя использовать для оценки более сложных индексов, таких, как распределение интервалов отказов. В общем случае моделью SLRUM следует пользоваться с осторожностью из-за того, что периоды самой напряженной страничной активности при любой стратегии соответствуют сменам локальных областей, которые она не способна воспроизвести должным образом.

Описание состояний программы, на котором базируется модель SLRUM, может использоваться в более сложных моделях. Например, можно построить марковскую модель, оценив вероятности переходов между расстояниями стека LRU. Их можно получить аналогично формуле (9.20) по счетчикам частоты всех пар (δ_i, δ_k) расстояний в строке расстояний. Матрица этих счетчиков частоты для строки из табл. 9.15 (a) с удаленными первыми одиннадцатью ссылками приводится в табл. 9.16 (c), а соответствующая матрица вероятностей переходов — в табл. 9.16 (d). Заметим, что эта модель, так же как и SLRUM, определена полностью только при задании $\mathbf{s}_0$. Таким образом, она состоит из пары ($\mathbf{s}_0$, $\{d_{ik}\}$), где $\{d_{ik}\}$ ($i, k=1, \ldots, u$) есть матрица вероятностей переходов расстояний.

Возможны и другие выборы состояний программы. Например, состояние можно идентифицировать с текущим стеком LRU, с текущим множеством ссылок или текущей локальной областью. Точное определение локальных областей программы является безусловным требованием при их использовании для описания поведения программ, что, конечно, является привлекательной идеей. Теперь мы кратко разберем эту проблему (что мы обещали сделать в разд. 9.2.2) на базе определения Мэдисона и Батсона (1976).

В любой момент времени, страницы текущей локальной области (при любом ее определении) имеют тенденцию перемещаться к вершине стека LRU. Пусть $\{i_j(t_j)\}$ — строка ссылок на страницы (или блоки, или сегменты), и пусть $\mathbf{s}_j$ — состояние стека LRU после j-й ссылки (причем $\mathbf{s}_0=0$). Заметим, что, если даже интервалы между последовательными ссылками в строке не постоянны, времена t_j можно игнорировать, когда мы предполагаем, что они известны для любого значения индекса строки j. Знание этого требуется только для определения продолжительности каждой локальной области во времени, а не для идентификации самих локальных областей. Пусть

$s_j(k)$ ($k=1, 2, \ldots, u$) — содержимое k-й позиции $\mathbf{s}_j$ и

$$Sh_j=(s_j(1), s_j(2), \ldots, s_j(h)) \tag{9.23}$$

— неупорядоченное множество содержимых h самых верхних позиций $\mathbf{s}_j$. Итак, для каждого стека имеется u множеств, являющихся кандидатами на роль текущей локальной области. Эти множества иерархически связаны друг с другом, так как

$$S1_j \subset S2_j \subset \ldots \subset Su_j. \tag{9.24}$$

Кроме того, каждое множество характеризуется своим *возрастом*, являющимся числом ссылок, для которых существовало это множество. Возраст Sh_j задается выражением $j+1-g_j(h)$, где $g_j(h)$ обозначает индекс строки, когда элементы Sh_j впервые появились в h самых верхних позициях стека. Другими словами, множество Sh_j генерируется при $g_j(h)$-й ссылке и с тех пор не меняется:

$$Sh_{g_j(h)} = Sh_{g_j(h)+1} = \ldots = Sh_j. \tag{9.25}$$

Каждый стек $\mathbf{s}_j$ обладает связанным с ним вектором $\mathbf{g}_j$, который содержит *даты рождений* множеств Sh_j ($h=1, \ldots, u$). Другой характеристикой множеств Sh_j является уровень его активности. Из интуитивного понятия локальной области следует, что страница должна быть частью текущей локальной области, если на нее ссылаются вместе с другими элементами этой области. В соответствии с этим Мэдисон и Батсон (1976) определяют Sh_j как *множество активности*, если на все его элементы ссылались хотя бы один раз с момента рождения этого множества. Самая первая из используемых страниц в Sh_j есть $s_j(h)$. Пусть $r_j(h)$ — значение индекса строки, соответствующее последней ссылке на $s_j(h)$. Если $r_j(h)>g_j(h)$, то Sh_j есть множество активности; в противном случае оно таким не является. Поэтому с вектором $\mathbf{s}_j$ будет связан еще один вектор $\mathbf{r}_j$, содержащий индексы строк самых последних ссылок на каждую страницу в порядке расположения страниц в стеке. Простое сравнение содержимых соответствующих элементов $\mathbf{r}_j$ и $\mathbf{g}_j$ позволит нам быстро определить все множества активности для любого j. Ясно, что данное определение множества активности не исключает существование различных таких множеств для каждого значения j. Эти множества иерархически взаимосвязаны как в терминах их элементов из-за (9.23), так и в терминах их времени жизни. *Время жизни* множества активности равно его возрасту в момент его смерти. Смерть множества активности Sh вызывается ссылкой, для которой стековое расстояние LRU больше h. Пример в табл. 9.17 показывает, что определить время смерти множества активности очень просто. Однако, прежде чем заняться анализом этого примера, мы должны привести правила построения и обновления векторов $\mathbf{s}_j$, $\mathbf{r}_j$ и $\mathbf{g}_j$. Правила для $\mathbf{s}_j$ были представлены в табл. 9.8, но для полноты изложения дадим их снова.

Таблица 9.17

Обнаружение множеств активности и ограниченных интервалов локальности в первых 32 ссылках строки из табл. 9.15 (а)

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
i_j	1	3	4	1	4	1	4	3	1	4	2	4	2	4	2	4	2	4	2	4	2	4	2	4	1	4	1	4	1	2	4	2
$\mathbf{s}_j$	1	3	4	1	4	1	4	3	1	4	2	4	2	4	2	4	2	4	2	4	2	4	2	4	1	4	1	4	1	2	4	2
	0	1	3	4	1	4	1	4	3	1	4	2	4	2	4	2	4	2	4	2	4	2	4	2	4	1	4	1	4	1	2	4
	0	0	1	3	3	3	1	4	3	1	4	2	4	2	4	1	1	1	1	1	1	1	1	1	2	2	2	2	2	4	1	1
	0	0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
δ_j	5	5	5	3	2	2	2	3	3	3	5	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3	2
	(∞)	(∞)	(∞)								(∞)																					
$\mathbf{r}_j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
	0	1	2	3	4	⑤	⑥	7	8	9	10	11	⑫	⑬	⑭	⑮	⑯	⑰	⑱	⑲	⑳	㉑	㉒	㉓	24	㉓	㉓	㉓	㉓	㉘	㉙	㉙
	0	0	1	2	2	2	⑥	⑦	⑧	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	8	8	8	8	8	8	8
	0	0	0	0	0	0	0	0	0	0	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
$\mathbf{g}_j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
	1	2	3	4	4	4	4	8	9	10	11	11	11	11	11	11	11	11	11	11	11	11	11	11	25	25	25	25	25	30	31	31
	1	2	3	3	3	3	3	3	3	3	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
	1	2	3	3	3	3	3	3	3	3	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11

Вначале мы имеем

$$\mathbf{s}_0 = \mathbf{r}_0 = \mathbf{g}_0 = \mathbf{0}. \tag{9.26}$$

Обозначим стековое расстояние LRU до следующей ссылки i_j через δ_j и, если i_j — новая страница, присвоим δ_j значение $u+1$ вместо ∞. Тогда для получения $\mathbf{s}_j$, $\mathbf{r}_j$, $\mathbf{g}_j$ по $\mathbf{s}_{j-1}$, $\mathbf{r}_{j-1}$, $\mathbf{g}_{j-1}$ нужно применять следующие правила:

$$\begin{aligned}
s_j(1) &\leftarrow i_j, \\
s_j(k+1) &\leftarrow s_{j-1}(k) && (k = 1, \ldots, \delta_j - 1), \\
s_j(k) &\leftarrow s_{j-1}(k) && (k = \delta_j + 1, \ldots, u), \\
r_j(1) &\leftarrow j, \\
r_j(k+1) &\leftarrow r_{j-1}(k) && (k = 1, \ldots, \delta_j - 1), \\
r_j(k) &\leftarrow r_{j-1}(k) && (k = \delta_j + 1, \ldots, u), \\
g_j(k) &\leftarrow j && (k = 1, \ldots, \delta_j - 1), \\
g_j(k) &\leftarrow g_{i-1}(k) && (k = \delta_j, \ldots, u).
\end{aligned} \tag{9.27}$$

В табл. 9.17 приведено содержимое трех векторов $\mathbf{s}_j$, $\mathbf{r}_j$ и $\mathbf{g}_j$ для первых 32 ссылок строки из табл. 9.15 (a). Обведенные окружностями элементы $\mathbf{r}_j$ — это элементы, значения которых больше соответствующих элементов $\mathbf{g}_j$. Поэтому каждый из них идентифицирует множество активности. Например, тот факт, что второй элемент $\mathbf{r}_6$ обведен окружностью, означает, что $S2_6$ — множество активности. Его элементами являются страницы в первых двух позициях $\mathbf{s}_6$ (π_1 и π_4). Такое же множество активности живет под именем $S2_7$ для $j=7$. Впрочем, так как $\delta_8 = 3 > 2$, это множество умирает при $j=8$. Дата его рождения задается значением $g_7(2)=4$. Следовательно, время его жизни равно 8—4=4 ссылкам. Конец строки обведенных окружностями элементов вектора $\mathbf{r}_j$ отмечает смерть соответствующего множества активности. Например, множество (1, 3, 4) формируется при $j=3$, становится множеством активности при $j=8$ и умирает при $j=11$, прожив 8 ссылок. Пара, состоящая из множества активности и длительности его жизни, например [(1, 3, 4), 8], называется *ограниченным локальным интервалом*.

На рис. 9.11 приводятся ограниченные локальные интервалы, которые можно выявить в строке из табл. 9.15 (a). Плотность ссылок для четырех страниц строки, вычисленная на интервалах в пять ссылок, показана на рис. 9.11 (a). Их взаимосвязь с ограниченными локальными интервалами иллюстрируется на рис. 9.11 (b) и совершенно очевидна. Заметим, что горизонтальные оси на этих диаграммах представляют не время, а число ссылок.

Как было сказано выше, существующая для множеств активности иерархия имеет место также и для ограниченных локальных интервалов. Те интервалы, у которых бо́льшие множества активности, никогда не обладают меньшим временем жизни. Кроме этого, поведение большинства программ [см. Мэдисон и Батсон (1976)], по-види-

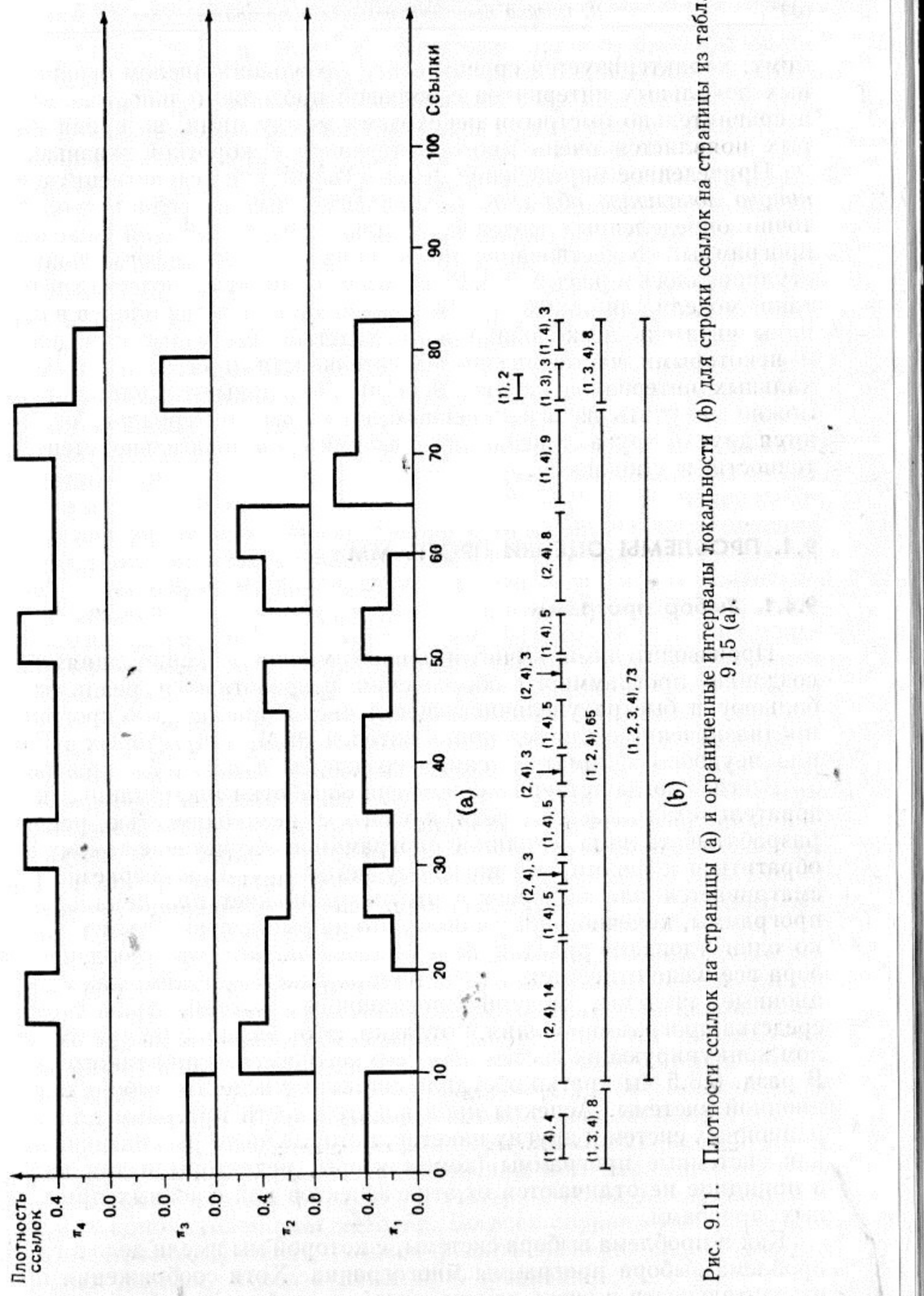


Рис. 9.11. Плотности ссылок на страницы (a) и ограниченные локальные интервалы локальности (b) для строки ссылок на страницы из табл. 9.15 (a).

мому, характеризуется сравнительно небольшим числом ограниченных локальных интервалов с большой продолжительностью жизни и сравнительно быстрыми переходами между ними, во время которых появляется очень много интервалов с короткой жизнью.

Приведенное определение не идентифицирует единственную *текущую локальную область*, а предлагает выбор среди нескольких точно определенных моделей локальных областей для поведения программы. Существование нескольких моделей неформально постулировалось в разд. 9.2.2. В табл. 9.6, например, представлены две такие модели для нашей строки, и после отображения блоков в страницы читатель легко обнаружит сходство между этими моделями и некоторыми частными последовательностями ограниченных локальных интервалов на рис. 9.11(b). Локальные модели, которые можно получить из ограниченных локальных интервалов, отличаются друг от друга степенью детализации и, следовательно, степенью точности и сложности.

9.4. ПРОБЛЕМЫ ОЦЕНКИ ПРОГРАММ

9.4.1. Выбор программ

Производителями вычислительных машин и корпорациями по созданию программного обеспечения разработано и реализовано большое и быстро увеличивающееся число прикладных программ, предназначенных для тех пользователей ЭВМ, для которых трудно или неудобно заниматься самим созданием нужных им программ.

Всегда, когда встает новая задача обработки информации, пользователь сталкивается среди прочего с необходимостью решить, разрабатывать ли необходимое программное обеспечение самому или обратиться к некоторому внешнему источнику. Если серьезно рассматривается или выбирается второе, возникает проблема выбора программы, конечно, при условии, что на рынке существует не только один удовлетворяющий пользователя пакет. Эта проблема выбора все чаще относится и к системным пакетам, таким, как операционные системы, модули операционных систем, трансляторы, средства программирования и отладки. Это связано с растущим числом конкурирующих систем программного обеспечения такого типа. В разд. 6.3.5 мы кратко обсудили системные аспекты выбора операционной системы. Аспекты производительности программ для операционных систем и других пакетов, которые часто рассматриваются как системные программы (компиляторы, редакторы и так далее) в принципе не отличаются от этих аспектов для сложных прикладных программ.

Как и проблема выбора системы, с которой мы имели дело в гл. 6, проблема выбора программы многогранна. Хотя соображения производительности и очень важны, сведение выбора одной из нескольких программ к сравнению их производительности было бы большой ошибкой. Функциональные и экономические факторы по крайней мере так же важны, как и соображения производительности, а часто и более важны. Впрочем, мы ограничимся в нашем кратком обсуждении только соображениями производительности.

Процедура выбора, описанная в разд. 6.2.1, может быть в принципе применена и для случая пакетов программного обеспечения. Но более подходящей для них кажется ее упрощенная версия, подобная той, которую мы обрисовали в конце разд. 6.2.1 для системных компонент. В частности, вместо того чтобы посылать запросы поставщикам, пользователи сами проводят сбор данных для контроля различных вариантов. Выполнение сравнения производительности пакетов-кандидатов на предполагаемой установке пользователя имеет то преимущество, что при этом устраняются все источники приблизительных оценок, которые были бы неизбежны при использовании другой установки или, еще более ясно, при использовании различных установок для различных программ. Было бы очень трудно определить, которая из двух программ P_1 и P_2 будет работать более эффективно на системе A, зная их времена выполнения t_{1B} и t_{2B} на системе B, особенно если эти времена не отличаются очень сильно друг от друга. Было бы еще хуже и действительно значительно хуже, если бы мы знали только времена t_{1B} и t_{2C}, измеренные для двух различных систем B и C. Сравнение производительности на системе пользователя является также самым убедительным способом проверки существенного функционального требования правильной работы рассматриваемого пакета на данной системе. Иногда проведение такого сравнения может оказаться невозможным. Продавец может не захотеть произвести установку пакета до того, как выбор сделан и контракт подписан, из-за связанного с этим риска или высокой стоимости модификаций или того и другого. Однако предполагаемый покупатель должен проявлять настойчивость, особенно в случае новых пакетов, которые еще не опробовались на различном оборудовании, или программ с различными оценками предыдущих пользователей.

Вне зависимости от того, делается ли сравнение на системе покупателя или на какой-то другой системе, по-видимому, не вызывает сомнений, что методом сравнения является измерение. Единственной альтернативой могла бы быть аналитическая оценка, если бы программа сопровождалась формулой или кривой, которая позволяла бы предсказывать время выполнения и другие представляющие интерес индексы для данной системы, данной конфигурации и конкретных исходных данных. Эти формулы или кривые могли бы выводиться непосредственно из измерений или аналитически так, как описано в примере 9.5, или получаться с помощью моделирования или комбинацией этих методов. Во всех случаях экспериментатор, оценивающий пакет, должен решить важную задачу выбора входных данных, используемых при сравнении.

Нельзя дать общих правил, помогающих сделать этот выбор. Если программы можно выполнять несколько раз в течение периода оценки, множества исходных данных должны покрывать типовую область входных данных, употребляемых пользователями пакета, который будет выбран. Среди них должно быть некоторое множество, представляющее реальное условие *тяжелой нагрузки*, т. е. входные данные, про которые известно или от которых ожидается, что они потребуют большого времени выполнения или системных ресурсов, которые значительно превосходят средние. Для сложных программ определение условия худшего ввода может оказаться крайне трудным или невозможным. Аккуратная оценка вероятностей появления различных входных множеств может быть легко выполнимой в одних случаях и невозможной в других. Так или иначе, учитывая, что число инструментированных прогонов ограничено, приоритет нужно отдавать тем множествам входных данных, которые, по мнению исследователя, могут создать критические ситуации. Например, при работе в реальном времени главной заботой должно быть максимальное значение времени выполнения, в соответствии с чем и должны разрабатываться тесты. Среднее время выполнения, по всей вероятности, более важно при пакетной обработке, т. е. в таких приложениях, как обработка платежных ведомостей, бухгалтерский учет, инвентаризация, производственное планирование и статистический анализ.

9.4.2. Усовершенствование программ

При оценке производительности программ, как и в случае систем, использование всех методов оценки более тщательно разработано и более широко распространено для целей усовершенствования, чем для любых других типов исследования. Измерение, имитационное моделирование и аналитические способы, которые мы рассмотрели в разд. 9.3.1 и 9.3.2, могут применяться как к проблемам диагностики, так и к проблемам прогнозирования результатов усовершенствования. Следует отметить, что здесь второе менее критично, чем при изучении производительности систем, так как модификация прикладной программы часто легче, дешевле и менее рискованна, чем модификация системы. Поэтому наиболее часто используемым методом оценки совершенствующих изменений является прямое измерение модифицируемой программы.

Как и в случае систем, диагностирование производительности программы имеет своей целью обнаружить причины, ограничивающие ее производительность, другими словами, ее *узкие места* или *ошибки производительности*. Ликвидация узких мест, как правило, улучшает производительность программы и снижает ее стоимость. В одних исследованиях больше внимания уделяется аспектам производительности, а в других — экономическим аспектам проблемы.

Впрочем, в любом случае достигаются обе цели, если будет сокращено потребление ресурсов системы.

Как можно обнаружить узкие места производительности программы? Другими словами, как определить, на каких участках программы в основном расходуется время ее выполнения? Эффективным средством диагностики является профиль программы, введенный в разд. 9.2.1. Он, безусловно, является более мощным средством, чем его системный эквивалент, профиль использования системы (см. разд. 2.7.1 и 7.3.3). Как описано ниже в примере 9.6, профиль программы с дополнительной информацией о времени выполнения в точности указывает координаты узкого места. Как знает каждый достаточно опытный программист, эту функцию обнаружения нельзя, как правило, оставлять автору программы, если мы хотим получить надежный результат. По-видимому, ум человека не способен справиться с уровнем сложности, которая присуща динамике большинства программ, если даже программы принадлежат категории непараллельных программ среднего размера.

Имея профиль программы, состоящий из счетчиков частоты c_i ($i=1, \ldots, n$) выполняемых операторов программы, можно определить *временной профиль* как объединение времен, затрачиваемых при выполнении каждого оператора. Если t_i означает время выполнения оператора i, то его вклад в общее время выполнения t есть $c_i t_i$ и, следовательно,

$$t = \sum_{i=1}^{n} c_i t_i. \tag{9.28}$$

Модификация программы может затрагивать несколько операторов. Их времена выполнения и счетчики частоты могут меняться очень сильно в результате даже единичного изменения, и может возникнуть потребность добавить или переместить некоторые операторы. Если мы рассмотрим самый простой из возможных типов модификации, тот, который влияет только на один оператор, скажем на оператор j, то из (9.28) мы имеем

$$\Delta t = c_j \Delta t_j, \tag{9.29}$$

так как из нашего предположения следует, что $\Delta c_j = 0$. Заметим, что формулу (9.29) можно легко обобщить на любую группу последовательно связанных операторов.

Временной профиль позволяет нам немедленно обнаружить узкое место программы, т. е. те операторы или группы операторов, которые дают наибольший вклад в значение t. Концентрировать усилия по усовершенствованию на этих областях программы удобно по следующим причинам:

а) Большое значение t_j может быть частично связано с потерями из-за небрежного программирования (ошибки производительности); поэтому может оказаться легче получить определенное (отри-

цательное) значение Δt_j или большее значение $|\Delta t_j|$, не затрачивая дополнительных усилий.

b) Любое Δt_j, которое мы получаем, далее умножается на c_j [см. (9.29)]; поэтому такие или меньшие усилия могут дать гораздо больший полезный результат по сравнению с их применением в других областях; очевидно, что этот эффект возрастает с ростом c_j.

c) Ограничение поля исследования на каждом этапе только несколькими операторами или некоторой подпрограммой обычно повышает качество получаемых результатов.

Конечно, когда изменение более радикально, чем только что рассмотренное, простые доводы из a и b, основанные на (9.29), уже не корректны, и следует учитывать также и изменения счетчиков частоты. Впрочем, в любом случае временной профиль очень полезен при идентификации изменений, к которым время выполнения программы будет наиболее чувствительно. Теперь мы проиллюстрируем его использование на примере.

Пример 9.6. Мы займемся анализом программы из табл. 9.1, чтобы определить, можно ли, и если да, то как, уменьшить время ее выполнения. Пусть ее профиль, полученный измерением (или даже с помощью имитационного моделирования или аналитическими методами, как описано в разд. 9.3.2), состоит из счетчиков частоты, которые приведены в той же самой таблице. Этот профиль имеет довольно нерегулярную структуру, которая была бы еще более нерегулярной, если бы входные данные были более реальными, так как это повлекло бы более частое выполнение циклов. Даже с теми входными данными, которые мы выбрали, три оператора (12, 13 и 14), составляющие немногим больше 10% всех выполнимых операторов, затрачивают 42% времени профиля.

В табл. 9.1 приведен также временной профиль той же пары (программа, входные данные). Больше всего времени использует оператор 12. Для его выполнения программе требуется примерно 18% ее времени. Функция FUN для своих трех операторов (12, 13, 14) поглощает примерно 35% общего времени выполнения. Эта ситуация не удивительна. На самом деле, как отмечено выше, эти профили более регулярны, чем обычно. Как правило, очень большая часть всего времени затрачивается на выполнение очень маленькой части программы. Кнут (1971) сообщает, что в программе на ФОРТРАНе в его экспериментах более половины всего времени обычно тратилось на выполнение менее 4% программы. Другой областью интенсивной активности является область между операторами 21 и 25, где программа тратит более 24% своего времени. Другие заслуживающие внимания операторы с достаточно высоким значением $c_i t_i$ это, по-видимому, только операторы 1, 3 и 9. Впрочем, программист, как правило, не имеет возможности изменить код программ ввода-вывода, вызываемых операторами ввода-вывода, такими, как READ и WRITE, а на время, затрачиваемое при обработке запросов ввода-вывода, может влиять только выбор типа ввода-вывода в программе. Решение этой проблемы зависит от системы и от компилятора. Только детальное знакомство со способом обработки операций ввода-вывода может позволить программисту пользоваться ими разумно, чтобы получить лучшие возможные результаты. В этом отношении организация нужных программе файлов и стратегии доступа к ним требуют основного внимания на этапе проектирования. Впрочем, часто эти параметры оказываются немодифицируемыми в связи с системными ограничениями. В нашем случае беглый просмотр операторов ввода-вывода достаточен, чтобы убедить нас, что мы почти ничего не можем сделать для их усовершенствования [за исключением, быть может, упаковки нескольких элементов вектора A(I) в строке перед выдачей на печать в операторе 9].

Вначале исследуем функцию FUN и особенно оператор 12. Время t_{12} тратится практически на установку связей, которые требуются всегда, когда обращаются к функции. Очевидный путь сокращения времени выполнения FUN — ликвидировать эту функцию и заменить каждое обращение к ней в подпрограмме RK2 соответствующей эквивалентной операцией, т. е. вычислением соответствующего эквивалента. Эта модификация для данной программы удобна в связи с простотой функции FUN и тем, что ее имя появляется в программе всего несколько раз. Однако во многих случаях такое изменение неудобно либо из-за увеличения объема программы, которое оно вызовет, либо из-за возможной демодуляризации, либо из-за того и другого.

Ликвидация функции FUN позволяет нам также произвести некоторые упрощения в операторах 21—25, т. е. в области нашей программы, которая, как мы установили, занимает по активности второе место. Оператор 23 можно удалить, так как он оказывается идентичным оператору 22. Без каких-либо больших усилий по оптимизации (мы могли бы, например, попытаться заменить деление в операторе 25 на умножение, если бы это оказалось выгодным) получаем следующие новые операторы:

```
21'   T1 = H/X
22'   T2 = H/(X + H2)
23'   —
24'   T4 = H/(X + H)
25'    Y = Y + (T1 + 4. * T2 + T4)/6.
```

Используя те же самые времена выполнения, как в табл. 9.1, и счетчики частоты из табл. 9.1, оцениваем получаемую в связи с этими изменениями экономию. Из общего времени 650 единиц экономится 278, что соответствует сокращению времени выполнения программы на 42%. Для более реальных исходных данных это сокращение было бы гораздо более значительным. Читатель может попытаться с помощью одной из аналитических моделей, разработанной для этой программы в разд. 9.3.2, оценить экономию для некоторых других множеств исходных данных (см. задачу 9.16). ■

Если программа написана на языке высокого уровня и изучается с целью ее усовершенствования на уровне этого языка, как, например, в примере 9.6, машина, которую представляет программист, есть машина языка высокого уровня. Внутренние характеристики этой машины должны быть хорошо известны программисту, если от исследования по усовершенствованию ожидается высокая вероятность успеха при наибольшей возможной полезной отдаче. Это означает, что исследователь должен хорошо разбираться в устройстве компилятора, особенно если это оптимизирующий компилятор. В противном случае внесенные исправления могут лишь внешне казаться эффективными, а на самом деле привести к снижению производительности программы. Влияние компилятора на производительность программы очень сильно. Использование языков высокого уровня является соединяющим фильтром между машинным оборудованием, которое в конечном счете определяет время выполнения программы, и программистами, которые уже не в состоянии оптимизировать свои программы применительно к этому оборудованию. Поэтому программа должна оптимизироваться в расчете на комплекс «машинное оборудование — компилятор». Эта менее прямая оптимизация, и накладываемые компилятором ограничения объясняют, почему полученные в результате трансляции с языков высокого уровня программы редко обладают производительностью,

которую бы они имели, если бы были написаны опытным программистом на языке машины, и почему некоторые программы, время выполнения которых очень важно, часто пишутся на языке машины или ассемблера.

Иногда усовершенствование касается расхода памяти. Эта проблема особенно существенна для программ, для которых объем памяти строго ограничен (например, резидентные части операционной системы или микропрограммы, хранимые в небольшой памяти управления). Эта проблема встречается также в связи с большими базами данных, когда прибегают к методам упаковки данных или экономным структурам данных. Для сокращения памяти, занимаемой командами программы, обрабатывают листинг программы, используя специфические методы.

В системах с виртуальной памятью очень важным с точки зрения производительности программы является сокращение числа отказов, генерируемых программой. В примере 7.10 мы видели, что это сокращение, как правило, увеличивает производительность системы. В разд. 9.2.2 мы убедились, что оно всегда увеличивает производительность программы и уменьшает стоимость ее выполнения. В разд. 9.2.2 также обсуждалось, что чем ближе поведение программы к поведению, постулируемому реализованной в системе стратегией управления памятью, тем ниже ожидаемая интенсивность отказов. Заметим, что эта точка зрения отправная для задачи усовершенствования программы, так как стратегия рассматривается как немодифицируемый объект, а изменять можно лишь поведение программы. В разд. 9.2.2 мы также узнали, что стратегии обычно используют свойство локальности. Отсюда можно заключить, что улучшение локальности программы, безусловно, увеличит ее производительность. Как мы увидим дальше, в принципе это верно. Однако верно и то, что каждая стратегия базируется на различных определениях локальности или, другими словами, на различных моделях поведения программы. Когда методы усовершенствования учитывают доступную информацию о стратегии, обычно получаются более успешные результаты. Эта методика получила название *подгонки программы* [см. Феррари (1975)], так как она пропагандирует методы усовершенствования, которые подгоняют поведение программы к поведению модели, постулируемому данной стратегией.

Как изменить динамическое поведение существующей программы, приблизив его к данной модели поведения? В некоторых ситуациях приемлемым ответом может быть разработка программы заново, но только если доступны методы разработки программ, которые соответствовали бы предписанному поведению. Ввиду того что так, к сожалению, не бывает, нужен более дешевый и более легкий в применении метод. Одним из таких методов является *программное переструктурирование*.

Метод программного переструктурирования предполагает, что поведение программы описывается строкой ссылок на блоки (см. разд. 9.2.2). Чтобы этот метод работал успешно, размер среднего блока должен быть существенно меньше размера страницы. Хорошее эмпирическое правило гласит, что он должен быть в интервале от 1/3 до 1/10 размера страницы. И вообще, чем меньше блоки, тем более эффективен, но и более дорог метод. Переструктурирование означает введение нового отображения этих блоков на страницы с целью изменения поведения программы в лучшую сторону. Заметим, что это отображение происходит на виртуальном уровне и должно применяться к распределению памяти для блоков в виртуальном пространстве. В нашем рассмотрении мы ограничимся случаем статического распределения памяти. Программа переорганизуется только один раз и затем выполняется, как если бы она была организована так самим автором. Удовлетворительный метод для динамического распределения блоков по страницам до сих пор еще не разработан. (Статическое) отображение, с которым мы имеем дело, не обязательно имеет что-либо общее с (динамическим) отображением виртуальных страниц на физические страницы, которое выполняется иерархией памяти. Впрочем, механизм, с помощью которого последнее реализуется, может с успехом учитываться при определении первого.

Различные отображения $B \to П$, где B — множество блоков, а П — множество страниц, преобразуют одну и ту же строку ссылок на блоки в различные строки ссылок на страницы, которые в общем случае будут продуцировать различные интенсивности отказов. Это не трудно понять, если заметить, что блоки, которые выбираются в память при выборе страницы, так же как и блоки, которые выбрасываются при замене страницы, меняются от отображения к отображению. Блоки, которые с большой вероятностью окажутся нужны вскоре после ссылки на некоторый блок и окажутся бесполезны после исчезновения надобности в этом блоке, следует группировать вместе с этим блоком. Это утверждение все же слишком расплывчато, чтобы служить удовлетворительной спецификацией алгоритма переструктурирования. Мы подробно опишем некоторые из таких алгоритмов после обсуждения типичных методов переструктурирования.

Некоторые из способов переструктурирования программ можно удобно описать в виде четырехшаговой процедуры, которую мы и представим в общем виде [см. Феррари (1974b)].

Шаг 1. Подлежащая переструктурированию программа разделяется на блоки, размер которых, как было установлено выше, существенно меньше размера страницы. Если блок совпадает с перераспределяемыми частями программы (подпрограммы, функции, процедуры, такие структуры данных, как массивы, деревья и т. д.), переструктурирование не влечет за собой перепрограммирования. Впрочем, это не обязательное требование, а лишь удобный критерий для выбора блоков. Сделав этот выбор, можно легко трансформиро-

вать трассировочную запись адресов, полученную при инструментированном прогоне программы, в соответствующую строку ссылок на блоки.

Таблица 9.18

Матрицы переструктурирования для примера 9.7

(a) Матрица близости [строка ссылок на блоки из табл. 9.4 (a)]

	2	3	4	5	6	7	8
1	17	—	—	1	—	2	3
2		—	—	1	—	2	1
3			16	4	4	—	—
4				—	—	—	—
5					30	—	—
6						—	—
7							—

(b) Матрица CLRU [строка из табл. 9.4 (a), $m=2$]

	2	3	4	5	6	7	8
1	8	—	—	2	1	4	4
2		—	—	2	1	2	3
3			—	8	8	—	—
4				4	4	—	—
5					5	—	—
6						—	—
7							1

(c) Матрица CWS [строка из табл. 9.4 (b), $\tau=2$ множества ссылок на блоки]

	2	3	4	5	6	7	8
1	—	2	2	2	2	1	2
2		2	2	2	2	1	2
3			—	1	1	—	1
4				1	1	—	1
5					—	—	2
6						—	2
7							2

Шаг 2. К строке ссылок на блоки применяется *алгоритм переструктурирования*. В результате этого шага получается матрица, называемая *матрицей переструктурирования* $\mathbf{R}=[r_{hk}]$. Эту матрицу можно также рассматривать как представление графа, называемого *графом переструктурирования*. Пример такого графа, соответствующего матрице из табл. 9.18(b), представлен на рис. 9.12.

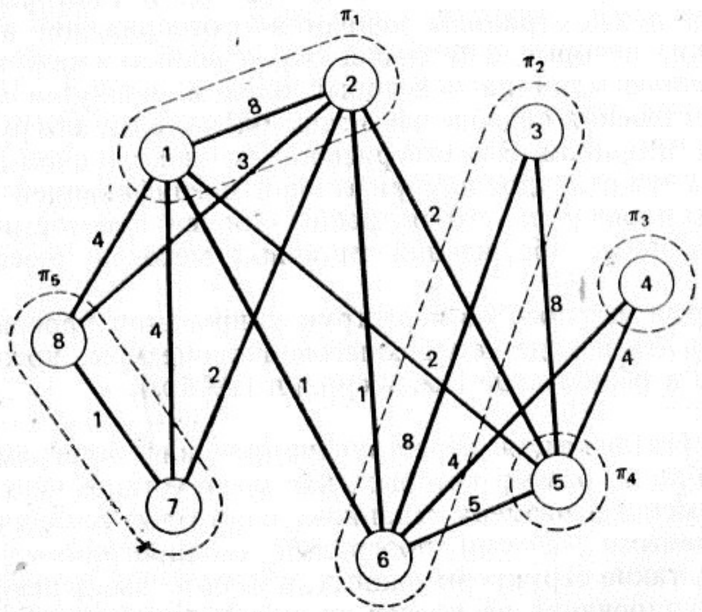


Рис. 9.12. Граф переструктурирования, соответствующий матрице CLRU из табл. 9.18(b). Оптимальное разбиение на группы выделено пунктирными линиями.

Вершины представляют блоки программы, каждая вершина имеет вес, равный размеру соответствующего блока. Хотя эта матрица может генерироваться алгоритмом как квадратная $u \times u$ матрица, где u — общее число блоков, на следующем шаге этой процедуры обычно работают с треугольной матрицей, элементы которой задаются формулой $r_{hk}+r_{kh}$. Эту матрицу можно рассматривать как представление неориентированного графа, такого, как граф на рис. 9.12. Метки ребер этого графа задают, в зависимости от используемого алгоритма переструктурирования, желаемость группировки в одной странице соответствующих двух вершин.

Шаг 3. К графу переструктурирования применяется *алгоритм разбиения на группы*. Решается следующая задача: объединить вершины графа в группы так, чтобы максимизировать сумму меток, внешних для групп ребер при условии, что сумма весов всех вершин в каждой группе не превышает размера страницы. Это хорошо известная задача из теории графов. Нахождение точного ее решения, т. е. оптимального отображения, не очень важно для успешной работы процедуры переструктурирования. Экспериментально был найден быстрый алгоритм разбиения на группы, который оказался настолько эффективным, что дальнейшие уточнения с помощью оптимальных алгоритмов, по-видимому, не принесут большой выгоды, но сделают вычисление гораздо более дорогим. Сложность любого алгоритма разбиения на группы растет очень быстро с увеличением числа вершин в графе. Поэтому это число нельзя стремиться увеличивать, учитывая только эффективность, связанную с уменьшением размера блоков.

Шаг 4. Группы вершин, определенные на шаге 3, соответствуют отображению блоков на страницы, являющемуся результатом описываемой процедуры. На этом шаге программа переорганизуется в виртуальной памяти согласно этому отображению. Размер группы обычно не совпадает с размером страницы. Поэтому, если информацию программы нужно скомпоновать так, чтобы ликвидировать некоторые (или все) неиспользованные промежутки в виртуальной памяти, для определения лучшего упорядочивания страниц на этом шаге можно применить еще один дополнительный алгоритм (возможно, также базирующийся на **R**). Если такой компоновки не делается, то порядок страниц, конечно, не существен.

Теперь мы опишем в качестве примеров три алгоритма переструктурирования, один из которых от стратегии не зависит, а два другие ориентированы на стратегию. При их пояснении мы будем обсуждать переструктурирование программы из табл. 9.1.

Пример 9.7. Описанная выше процедура переструктурирования будет применена к программе из табл. 9.1 в предположении, что программа должна выполняться при двух стратегиях управления памятью со страничной обработкой по требованию:

a) стратегия фиксированного распределения памяти с алгоритмом замещения LRU и выделенным пространством из $m=2$ физических страниц (*стратегия LRU*);

b) стратегия рабочего множества с переменным распределением памяти, размер

окна $\tau=50$ единицам времени, измерение рабочего множества и обновление производятся каждые 25 единиц времени (кратко, *стратегия WS*).

На шаге 1 программа разделяется на блоки. Мы выбираем блоки, перечисленные в табл. 9.3, которые являются перемещаемыми по отношению друг к другу на уровне исходного языка. Строки ссылок на блоки — это строки из табл. 9.4 (a) для стратегии LRU (которая не учитывает времена ссылок) и из табл. 9.4 (b) для стратегии WS. Для простоты мы предполагаем, что все блоки имеют размер в половину страницы, за исключением блока 5, размер которого совпадает (или немного меньше) с размером страницы. Таким образом, программа будет состоять по крайней мере из 5 страниц, хотя не все из них будут заполнены.

На шаге 2 требуется с помощью алгоритма переструктурирования получить матрицу переструктурирования. Простым, не зависящим от стратегии алгоритмом является *алгоритм близости*, предложенный Хэтфилдом и Джеральдом (1971). В результате получается *матрица близости*, которая совпадает с матрицей счетчиков частоты переходов, введенной в разд. 9.3.2 [см. табл. 9.16 (a)]. Заметим, однако, что матрица близости получается по строке ссылок на блоки (а не на страницы) и ее элементы r_{hk} — это значения количества переходов от блока h к блоку k. Другими словами, желательность объединения блоков h и k оценивается значением $r_{hk}+r_{kh}$, т. е. тем, сколько раз они встречаются вместе в строке ссылок на блоки. Треугольная матрица близости для нашей строки приводится в табл. 9.18 (a). Заметим, что эта матрица не зависит от времен ссылок.

Теперь попытаемся придумать другие алгоритмы переструктурирования с помощью метода подгонки. Обе стратегии, которые мы здесь рассматриваем, позволяют нам по соответствующей строке ссылок на блоки определить блоки, которые будут в памяти в любой заданный момент времени. Нам также нужно решить, какова специфическая цель этих алгоритмов. Пусть для обоих случаев это будет минимизация числа страничных отказов. Модели поведения программ, постулируемые обеими стратегиями, могут быть охарактеризованы минимальной вероятностью отсутствия в памяти следующей используемой страницы для любого момента времени. Находящиеся в памяти страницы занимают в этот момент первые m позиций стека LRU (если используется стратегия LRU) или являются элементами рабочего множества на этот момент (если используется стратегия WS). В любом случае эти страницы, безусловно, включают блоки, которые находятся в этот момент либо в верхних m позициях

Таблица 9.19

Алгоритм переструктурирования CLRU

Ввод: l = длина строки ссылок на блоки
u = число блоков в программе (размеры можно не учитывать)
m = число физических страниц, выделенных программе
$\{i_j\}$ = строка ссылок на блоки (времена ссылок можно не учитывать)
Вывод: $[r_{hk}]$ = матрица CLRU ($h,k=1, \ldots, u$)

Алгоритм:

1. (Присвоить начальные значения стеку блоков LRU, указателю на текущую ссылку, матрице CLRU.) Установить $s_h \leftarrow 0$ для $h=1, \ldots, u$, $j \leftarrow 0$, $r_{hk} \leftarrow 0$ для h, $k=1, \ldots, u$.
2. (Перейти к следующей ссылке и на вершину стека.) Установить $j \leftarrow j+1$, $n \leftarrow 0$.
3. (Найти в стеке текущую ссылку.) Установить $n \leftarrow n+1$. Если $n > u$, установить $\delta \leftarrow u$ и перейти к 4. Если $s_n=0$, установить $\delta \leftarrow n$ и перейти к 4. Если $s_n=i_j$, установить $\delta \leftarrow n$ и перейти к 4. В противном случае повторить этот шаг.
4. (Обновить матрицу CLRU.) Если $\delta > m$, установить $h \leftarrow i_j$, $k \leftarrow s_n$, $r_{hk} \leftarrow r_{hk}+1$ для $n=1, \ldots, m$ и для $s_n \neq 0$.
5. (Обновить стек.) Установить $s_h \leftarrow s_{h-1}$ для $h=\delta, \delta-1, \ldots 2$, $s_1 \leftarrow i_j$.
6. (Проверить условие окончания.) Если $j < l$, перейти к 2. В противном случае окончание.

стека блоков, либо соответственно в рабочем множестве блоков. Если следующий используемый блок в любом случае будет в памяти или из-за того, что его расстояние в стеке блоков не превосходит m, или из-за того, что он в рабочем множестве блоков, то говорят, что ссылка *некритическая*, и ничего не делается. Если ссылка *критическая*, то отображение должно сделать ее некритической, т. е. подготовить соответствующий блок в памяти к моменту ссылки. Чтобы поведение программы лучше соответствовало поведению модели, блок, упоминаемый в критической ссылке, должен группироваться по крайней мере с одним из блоков, уже находящихся в памяти в момент генерации ссылки. Из-за обычно возникающих конфликтных ситуаций нам необходимо сделать выбор такой группировки, которая бы наиболее резко сокращала число страничных отказов. Для этого мы можем, например, увеличить на 1 метку каждого ребра, соединяющего вершину, которая представляет блок, упоминаемый в критической ссылке, с вершинами, представляющими блоки, которые с большой вероятностью должны находиться в памяти в это время. Например если такими блоками являются блоки 1, 2 и 7, а следующая ссылка производится на блок 8, то метки ребер (8,1), (8,2) и (8,7) увеличиваются на 1. Это соображение является рациональным зерном класса алгоритмов переструктурирования, которые называются *алгоритмами критических множеств* [см. Феррари (1974а)]. Оба алгоритма этого класса, которые подгоняют программы к рассматриваемым двум стратегиям, подробно представлены в табл. 9.19 (*критический* LRU- или CLRU-алгоритм) и в табл. 9.20 (алгоритм *критического рабочего множества*, или CWS-алгоритм). *Матрица CLRU* и *матрица CWS*, полученные этими алгоритмами для нашей программы, приводятся соответственно в табл. 9.18 (b) и (c).

Алгоритм разбиения на группы, который не зависит от выбранного алгоритма переструктурирования, затем применяется к графу переструктурирования. В нашем простом примере легко видеть, что отображение из табл. 9.21 является лучшим для

Таблица 9.20

Алгоритм переструктурирования CWS

Ввод: l = длина строки ссылок (выражаемая числом множеств ссылок на блоки)
u = число блоков в программе (размеры можно не учитывать)
$\{b_j\}$ = строка множеств ссылок на блоки (множества ссылок на блоки во время интервалов равной длительности)
l_j = число блоков в множестве ссылок на блоки $b_j=\{b_{jk}\}$ ($j=1, \ldots, l$; $k=1, \ldots, l_j$).
τ = размер окна (число множеств ссылок)
Вывод: $[r_{hk}]$ = матрица CWS ($h,k=1, \ldots, u$)
Алгоритм:

1. (Присвоить начальные значения указателю на текущее множество ссылок, счетчикам блоков в окне, матрице CWS.) Установить $j \leftarrow 0$, $c_i \leftarrow 0$ для $i=1, \ldots, u$, $r_{hk} \leftarrow 0$ для h, $k=1, \ldots, u$.
2. (Перейти к следующему множеству ссылок на блоки.) Установить $j \leftarrow j+1$, $n \leftarrow l_j$; $k \leftarrow 0$.
3. (Перейти к следующей ссылке в множестве ссылок и определить, не является ли она критической.) Установить $k \leftarrow k+1$. Если $k > n$, перейти к 5. В противном случае установить $i \leftarrow b_{jk}$. Если $c_i=0$, перейти к 4. В противном случае повторить этот шаг.
4. (Обновить матрицу CWS.) Для $h=1, \ldots, u$, если $c_h \neq 0$, установить $r_{ih} \leftarrow r_{ih}+1$. Перейти к 3.
5. (Обновить счетчики.) Для $k=1, \ldots, n$ установить $i \leftarrow b_{jk}$, $c_i \leftarrow c_i+1$. Если $j \leqslant \tau$, перейти к 6. В противном случае установить $n \leftarrow l_{j-\tau}$. Для $k=1, \ldots, n$ установить $i \leftarrow b_{j-\tau,k}$, $c_i \leftarrow c_i-1$.
6. (Проверить условие окончания.) Если $j < l$, перейти к 2. В противном случае окончание.

трех матриц из табл. 9.18 при данных ограничениях. Пунктирные линии на рис. 9.12 соответствуют группам, построенным по матрице CLRU из табл. 9.18 (b). Заметим, что отображение, определяемое алгоритмом CWS (см. табл. 9.21), является одним из нескольких оптимальных разбиений на группы для матрицы CWS из табл. 9.18 (c).

Производительность переструктурированной программы можно определить с помощью имитационного моделирования, как описано в разд. 9.3.2, после преобразования строки ссылок на блоки в соответствующие строки ссылок на страницы согласно отображению, задаваемому процедурой переструктурирования. Результаты для трех переструктурированных версий одной и той же программы показаны в табл. 9.21. Число страничных отказов, генерируемых для стратегии LRU версией, соответствующей алгоритму CLRU, наименьшее из трех. Это также верно для отображения CWS и стратегии WS. Заметим, что, как и следовало ожидать, алгоритм CWS сокращает число страничных отказов, но в то же самое время имеет тенденцию к увеличению размера среднего рабочего множества по сравнению с другими алгоритмами. Впрочем, если непереструктурированная программа недостаточно близка к оптимальной, имеется алгоритм CWS, существенно сокращающий размер среднего рабочего множества по отношению к переструктурируемой программе. Целью CWS является не минимизация этого размера. Фактически, алгоритм близости и другие часто дают программы с меньшими размерами среднего рабочего множества. Алгоритмы переструктурирования, которые нацелены на минимизацию размера среднего рабочего множества, легко получить при использовании метода подгонки и рассмотрении этого размера как индекса производительности.

Две картины нашей программы на рис. 9.13 наглядно представляют некоторые из результатов переструктурирования. На горизонтальных осях отмечены не времена, а ссылки, как и на рис. 9.5. Верхняя картина — это картина строки ссылок на страницы из табл. 9.15 (a). Нижняя картина представляет строку, полученную алгоритмом CLRU. Обе диаграммы должны помочь читателю понять, почему последняя строка дает при стратегии LRU меньше страничных отказов, чем первая, даже при условии, что соответствующая программа состоит не из четырех страниц, а из пяти.

Таблица 9.21

Отображения, согласно матрице из табл. 9.18, и соответствующие индексы производительности

	Номер страницы	Блоки в страницах: Близость	CLRU	CWS
Отображения	1	1,2	1,2	1,3
	2	3,4	3,6	2,4
	3	5	4	5
	4	6	5	6
	5	7,8	7,8	7,8
Количество страничных отказов, генерируемых при LRU, когда $m=2$ страницам		18	14	31
Количество страничных отказов, генерируемых при стратегии WS, когда $\tau=2$ множествам ссылок на блоки		7	7	6
Средний размер рабочего множества при стратегии WS, когда $\tau=2$ множествам ссылок на блоки [страницы]		2.82	2.82	3.76

Эксперименты, проведенные с разными программами, показали, что, как и следовало ожидать, степень усовершенствования, получающегося в результате применения описанной выше процедуры переструктурирования, связана с качеством поведения непереструктурированной программы. Эта процедура, даже если она основывается на неоптимальных алгоритмах переструктурирования и разбиения на группы, обычно дает лучшие результаты для худших программ. Ее эффективность зависит от степени отличия поведения данной программы от ее неизвестной оптимальной версии. Получающееся в результате работы описанной процедуры отображение часто можно считать почти оптимальным. Дальнейшее его улучшение обычно оказывается очень трудным и недостаточно целесообразным.

Даже если в связи с плохой локальностью первоначальной программы ее можно значительно усовершенствовать, стоимость процедуры переструктурирования не оправдывается, когда программу собираются выполнять всего несколько раз. Все типы исследований по усовершенствованию, описанные в этом разделе, следует рассматривать только для программ, число прогонов которых больше некоторого минимума. В случае улучшения локальности посредством переструктурирования этот минимум зависит главным образом от качества программы, от конкретно используемого алгоритма переструктурирования и от выбора блоков. Для всех производственных программ, составляющих большую часть рабочей нагрузки на

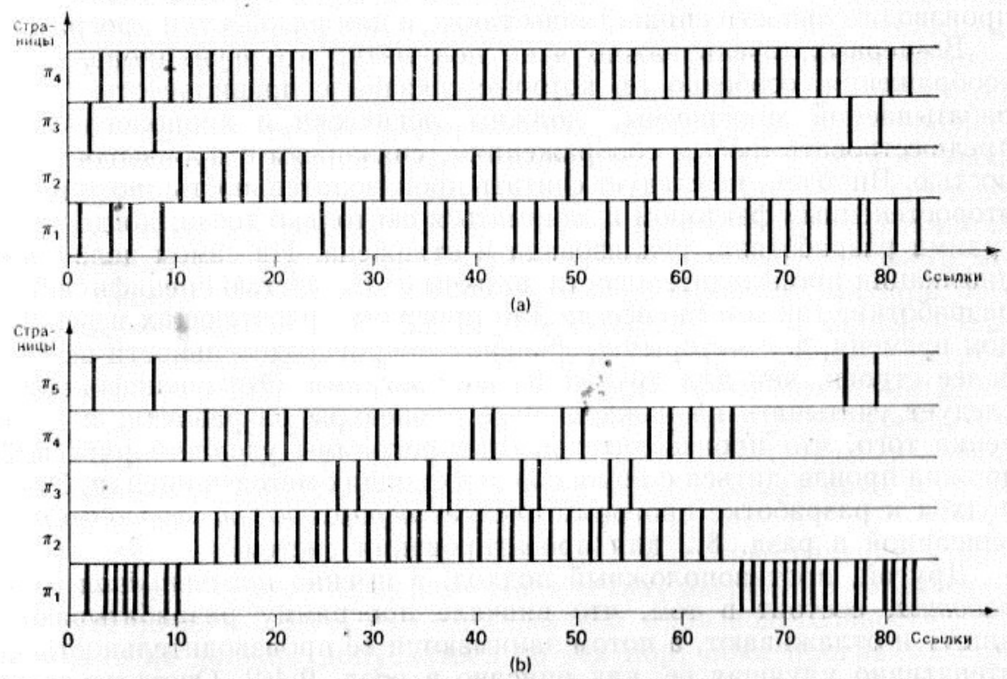


Рис. 9.13. Картины программы из табл. 9.1, полученные по (a) строке ссылок на страницы из табл. 9.15 (a), (b) по строке, полученной по отображению, соответствующему алгоритму переструктурирования CLRU (см. табл. 9.21).

большинстве систем, переструктурирование, безусловно, выгодно. Заметим также, что за точкой окупаемости вся экономия каждого прохождения включается в общий баланс, так как процедура применяется, как правило, только один раз.

По-видимому, самое серьезное возражение против описанной выше процедуры связано с ролью входных данных. Алгоритм переструктурирования применяется к строке ссылок на блоки, которая фактически представляет поведение пары (программа, входные данные). Если поведение программы очень чувствительно к входным данным, то отображение, задаваемое процедурой переструктурирования, с данными, отличными от данных инструментированного прогона, может отрицательно повлиять на поведение переструктурированной программы. Впрочем, различные эксперименты показали, что поведение программ, таких, как компиляторы, ассемблеры, редакторы, операционные системы и другие производственные программы, для которых переструктурирование очень выгодно, удивительно нечувствительно к входным данным по крайней мере с точки зрения эффективности переструктурирования [см., например, Хэтфилд и Джеральд (1971) и Феррари (1974)].

9.4.3. Разработка программ

Большинство замечаний, сделанных в разд. 8.2, по поводу общих вопросов проектирования систем и о роли методов оценки их производительности справедливы также и для разработки программ.

Во-первых, очень важно ясно понимать, что функциональные соображения, особенно те, которые связаны с правильностью разрабатываемой программы, должны логически и хронологически предшествовать любым соображениям, связанным с производительностью. Впрочем, не следует считать производительность программы второстепенным фактором и заниматься ею только тогда, когда программа разработана, реализована и отлажена. На самом деле спецификации производительности должны стать частью спецификаций разработки; так и было всегда для программ, работающих в реальном времени, для которых требования к производительности всегда более строги, чем для других типов программ. Эти спецификации следует учитывать на каждом шаге процедуры разработки, и проверка того, что производительность программы удовлетворяет им, должна производиться с помощью подходящих методов оценки. Этот подход к разработке программ следует *иерархической методологии*, описанной в разд. 8.2 для проектирования систем.

Другой, противоположный подход, а именно *итеративная разработка*, состоит в том, что вначале программу разрабатывают, пишут и отлаживают, а потом занимаются ее производительностью, итеративно улучшая ее, как описано в разд. 9.4.2. Очевидно, что самым важным (а возможно, и единственным пригодным) методом оценки при итеративной разработке является измерение (см.

разд. 9.3.1). Для улучшения локальности можно прибегнуть к смешанным методам измерения и моделирования, которые мы назвали процедурой переструктурирования. Большинство программ, работающих не в реальном времени, в настоящее время разрабатываются с помощью методов, которые ближе к итеративным, чем к иерархическим. Как и при проектировании систем, нужно стремиться к компромиссу между получаемой выгодой и стоимостью попыток итеративного усовершенствования. Другими словами, стоимость разработки может быть увеличена для сокращения стоимости выполнения. Оптимальный компромисс зависит от различных факторов, включая используемый язык (выбор которого оказывает нетривиальное влияние на производительность программы, связанное с процессом компиляции), стоимость и простоту измерения и модификации программы, ожидаемую продолжительность жизни программы и, очевидно, качество исходного варианта, которое определяется способностями разработчиков и программистов.

Иерархический подход характеризуется, как описано в разд. 8.2 и 8.4, последовательными уровнями абстракции, для генерации которых идеально подходит метод обработки сверху вниз. На каждом уровне абстракции определяется некоторая модель программы, обычно в терминах количества модулей и их взаимосвязей. На следующем уровне эти модули должны функционально подразделяться на подмодули. Предполагается, что определены требования производительности к каждому модулю. Например, пусть главным индексом производительности будет максимальное время выполнения. Когда мы разделяем модуль, максимальное время выполнения которого известно, мы должны оценить максимальное время выполнения каждого подмодуля. Из требований производительности на уровне i нам нужно уметь выводить требования производительности на уровне $i+1$. Сделав это, мы, прежде чем идти дальше, должны удостовериться, что если подмодули удовлетворяют своим спецификациям производительности, то требования производительности самого модуля также будут удовлетворяться. Качество результата этой процедуры, основанной на оценках, соответствует аккуратности выбранных оценок. При достижении самого низкого уровня абстракции, как правило, приходится пересматривать оценки и учитывать соответствующие изменения вплоть до самой вершины дерева. Таким образом, также из-за соображений производительности описываемая процедура обычно объединяет методы сверху вниз и снизу вверх.

Необходимая в этом случае проверка, как при создании нового уровня, так и в течение любого этапа «снизу вверх» при разработке, требует использования методов анализа производительности. Измерения не должны допускаться до тех пор, пока программа полностью или хотя бы частично не закодирована и не начинает работать. Поэтому будут использоваться имитационное моделирование или аналитические методы, как описано в разд. 9.3.2. Теперь мы

дадим иллюстрацию использования аналитического метода, рассмотренного в примере 9.5, для очень простой ситуации.

Пример 9.8. Последовательные Д-карты, которые были получены в примере 9.5 по карте с рис. 9.9 (а) для сведения ее к одному прямоугольнику, можно рассматривать как модели программы из табл. 9.1 на различных уровнях абстракции. Уровень соответствует шагу в табл. 9.11 (а). В процедуре «сверху вниз» уровень 0 соответствует шагу 6. На этом уровне у нас имеется прямоугольник X. Предположим, что максимальное время выполнения X есть $TX_{\max}$. Очевидно, что $TX_{\max}$ — это максимальное время выполнения всей программы. X разделяется на три прямоугольника A, W и M при переходе с уровня 0 на уровень 1. Поэтому

$$TX_{\max} = TA_{\max} + TW_{\max} + TM_{\max}. \tag{9.30}$$

В данной программе гораздо легче оценить $TA_{\max}$ и $TM_{\max}$, чем $TW_{\max}$. Поэтому оценку $TW_{\max}$ мы получим из (9.30) после оценки двух других величин. Если мы сделаем это, то условие, чтобы требования уровня 0 удовлетворялись при удовлетворении требований уровня 1, выполняется автоматически из-за нашего использования (9.30) для спецификации $TW_{\max}$. Заметим, что, так как из первого уравнения профиля в табл. 9.12 следует, что A, W и M последовательно связаны, и так как мы знаем, что $NX=1$, времена выполнения этих трех прямоугольников при каждом входе в них совпадают с соответствующими общими временами выполнения, которые только что были оценены:

$$ta_{\max} = TA_{\max},\ tw_{\max} = TW_{\max},\ tm_{\max} = TM_{\max}. \tag{9.31}$$

Уровень 2 получается при замене W итерацией, в которой появляется прямоугольник V. Поэтому, мы можем написать

$$TV_{\max} = TW_{\max} = nv_{\max}(1) \cdot tv_{\max}. \tag{9.32}$$

Для вычисления $tv_{\max}$, т. е. требования производительности на уровне 2, нужно оценить $nv_{\max}(1)$. Здесь нам должно помочь наше знание функциональных аспектов разрабатываемой программы. В данном случае nv (1) есть число значимых входных карт, и не представит труда установить разумную верхнюю границу для nv (1). В некоторых случаях может иметь смысл обратиться к вероятностной оценке. Оцененное значение $tv_{\max}$ далее нужно разбить на три слагаемых $tr_{\max}$, $tu_{\max}$ и $tt_{\max}$, сумма которых должна равняться $tv_{\max}$. Эту процедуру следует повторять, пока не дойдем до уровня 6 [соответствующего шагу 0 в табл. 9.11 (а)]. Каждый раз при выделении листа дерева для него можно написать код и с достаточной точностью вычислить или измерить его время выполнения. Далее следует немедленно отослать эти результаты обратно к корню дерева и сделать необходимые поправки оценок на различных уровнях. Например, код для A и M можно написать на уровне 1 и использовать для вычисления $ta_{\max}$ и $tm_{\max}$. ■

Практические программы будут в большинстве случаев гораздо сложнее программы, обсуждаемой в примере 9.8. При разработке больших систем программного обеспечения количество составляющих модулей может оказаться очень большим, а их типы и взаимосвязи более сложными. Впрочем, если они разрабатываются в соответствии с принципами структурного программирования, применение аналитических или имитационных методов или их объединения не слишком трудно. Но проблема, которую приходится решать всегда, это проблема определения множества входных данных для анализа производительности программы. Например, если главная спецификация производительности есть максимальное время выполнения, то мы должны знать или как-то догадаться, какое множество входных данных среди возможных максимизирует время выполнения.

Использование только что описанной иерархической методологии становится все более популярным, главным образом в задачах, связанных с правильностью программы. Введение в эту методологию аспектов производительности возможно, когда выбираются такие индексы, как время выполнения. Проверка ссылочного поведения во время иерархической разработки в настоящее время, по-видимому, представляет значительно больше трудностей. Однако вполне можно надеяться, что методы структурного программирования благотворно повлияют на программы в отношении их локальности; это связано с критерием разбиения модулей при применении этих методов.

В той степени, в какой дело касается ориентированной на производительность разработки программ, у нас еще нет систематической методологии для синтеза программ непосредственно по спецификациям производительности. Однако результаты анализа многих алгоритмов, представляющих практический интерес, позволяют программистам выбирать эффективные вычислительные методы и структуры данных в ряде случаев, которые часто встречаются при разработке программ [см., например, Кнут (1969)]. Ориентированный на локальность анализ алгоритмов привлек внимание исследователей позднее, и соответствующих результатов немного. Требуется более глубокое изучение связи структуры программы с ее поведением.

Заключение о необходимости дальнейших исследований встречается в нескольких местах этой книги, но, возможно, не так часто, как следовало бы. Наверное, это наименее дискуссионное утверждение, касающееся положения дел во многих областях оценки производительности. Все эти области требуют вдумчивого отношения и творческого вклада наших читателей.

Задачи

9.1. Повторите рассуждения из начала примера 9.1 о возможных определениях времени выполнения, применив их к программе, работающей на диалоговой системе. Перечислите некоторые из возможных определений и объясните их преимущества и недостатки с точек зрения системной независимости, отношения к пользователю и измеряемости.

9.2. Программа из табл. 9.1 будет выполняться на системе с пакетной обработкой. Формула для стоимости выполнения шага задания на этой установке следующая:

$$\begin{aligned}\text{JSPRICE} = {} & \text{CPUTIME} * \text{CPURATE} + \text{ETIME} * \text{ERATE} + \\ & + \text{DKOPS} * \text{DKRATE} + \text{CARDS} * \text{CRATE} + \text{LINES} * \text{LPRATE},\end{aligned}$$

где

CPUTIME = потребляемое время ЦП в секундах
CPURATE = стоимость секунды использования ЦП в долларах
ETIME = (CPUTIME+(DKOPS+CARDS+LINES)*IOTIME)*SPACE
ERATE = стоимость фактического времени в долларах за K байтосекунд
DKOPS = число операций с диском
DKRATE = стоимость использования диска ввода-вывода в долларах за одну операцию с диском

CARDS = число карт ввода
CRATE = стоимость использования устройства чтения карт ввода в долларах за карту
LINES = число строк печати
LPRATE = стоимость использования печатающего устройства вывода в долларах за строку
IOTIME = приблизительное время в секундах на одну операцию ввода-вывода
SPACE = объем занимаемой памяти в K байтах

Рассматривая только выполнение шага задания, определите связь между временем выполнения и стоимостью выполнения программы. В предположении, что объектный код и исходные данные уже хранятся на диске перед началом выполнения шага задания, выберите (специфицируйте точно) определение времени выполнения, которое вам нравится. Как лучше всего охарактеризовать программу в терминах состояний и соответствующей декомпозиции общего времени выполнения, чтобы решить эту задачу?

9.3. Постройте модель ФОРТРАН-ресурсов для программы из табл. 9.1 на базе типов ресурсов, перечисленных в табл. 6.9. Воспользуйтесь процедурой из табл. 6.11. Постройте также синтетическую модель программы, как показано в примере 6.3.

9.4. Является ли для программы трассировочная запись адресов (определенная в разд. 9.2.2) действительно полной характеристикой, по которой можно вычислить индекс поведения программы, если в качестве этого индекса для данной стратегии мы выберем общее число страниц обмена между основной памятью и устройством страничной обработки, а не общее число страничных отказов? Если ваш ответ отрицательный, сформулируйте, как бы вы ее модифицировали, чтобы вычисление этого индекса стало возможно.

9.5. При стратегиях рабочего множества исключенная из рабочего множества страница не обязательно сразу же выбрасывается из основной памяти. Соответствующее место в физической памяти присоединяется к области свободной памяти, но его содержимое может сохраняться до тех пор, пока оно не потребуется одной из программ. Поэтому, когда на исключенную из рабочего множества страницу снова делается ссылка через не слишком большой промежуток времени, можно надеяться, что она все еще находится в памяти. В этом случае она может быть *перезапрошена*, что исключает необходимость обращения к устройству страничной обработки. Можно ли как-то учесть эти соображения при обработке строки ссылок на страницы для вычисления числа генерируемых программой страничных отказов? Можно ли вычислить точное число отказов в этом случае? Как могли бы мы решить эту задачу статистически? Каков был бы смысл числа отказов, полученного алгоритмом из табл. 9.9, который не учитывает возможности перезапроса страниц?

9.6. Инструментируйте программу из табл. 9.1 так, чтобы можно было измерить прямо или косвенно (т. е. вычислить по другим измеренным величинам) время, затраченное в подпрограмме RK2 и функции FUN. Определите аналогично тому, как это сделано в табл. 9.7, необходимые изменения в тексте программы.

9.7. Как бы вы измерили условную вероятность перехода от одной вершины графовой модели программы к другой вершине? Например, покажите, как бы вы инструментировали программу из табл. 9.1 для измерения вероятностей перехода от вершины I к вершине II и от V к III в графовой модели на рис. 9.6 (b).

9.8. Предполагая, что ссылки в строке из табл. 9.4 (a) генерируются через правильные интервалы времени, вычислите число блочных отказов для программы при стратегии LRU. Чтобы получить за один просмотр строки кривую времени жизни программы (для блоков равного размера), алгоритм из табл. 9.8 нужно модифицировать. Размер памяти m неизвестен, и одну переменную nf нужно заменить u счетчиками $nf_1, nf_2, \ldots, nf_u$, в которых накапливается число генерируемых отказов при значении m, равном соответственно 1, 2, . . . u. Увеличение значений этих счетчиков при каждой исследуемой ссылке произойдет или нет в зависимости от значения δ для этой ссылки. Нарисуйте кривую времени жизни, которую вы вычислили с помощью модифицированного алгоритма.

9.9. В тех же предположениях, что и в задаче 9.8, примените алгоритм из табл. 9.9 к строке из табл. 9.4 (a) для вычисления числа блочных отказов и размера среднего рабочего множества $\bar{w}$ (в числе блоков), которые генерировались бы этой программой при стратегии чистого рабочего множества для $\tau=2, 3, 4, 5$ и 6 блоков. Нарисуйте диаграммы для $\bar{e}$ и $\bar{w}$ относительно τ, аналогичные диаграммам на рис. 9.10 (b). Если вы решили задачу 9.8, сравните производительности этой программы при стратегии LRU и стратегии рабочего множества, считая $\bar{w}$ индексом запроса памяти и сравнивая диаграмму $\bar{e}$ по $m=\bar{w}$ с кривой времени жизни, полученной в задаче 9.8. Попытайтесь объяснить несоответствия обеих кривых; являются ли эти результаты, по вашему мнению, нетипичными (т. е. присущими данной программе) или имеют более общий смысл.

9.10. Повторите вычисление среднего времени выполнения программы из табл. 9.1, основанное на вероятностной модели, которая описана в примере 9.4, для следующих значений вероятностей ветвления:

(a) $p_{hi}=0.8$, $p_{lj}=0.8$, $p_{ni}=p_{rq}=0.9$;
(b) $p_{hi}=0.9$, $p_{lj}=0.7$, $p_{ni}=p_{rq}=0.9$;
(c) $p_{hi}=0.9$, $p_{lj}=0.8$, $p_{ni}=p_{rq}=0.8$.

Какие выводы относительно чувствительности среднего времени выполнения к этим вероятностям можете вы сделать по этим результатам? Не смогли бы вы упорядочить эти вероятности по их влиянию на среднее время выполнения, опираясь только на модель программы с рис. 9.7 (a)?

9.11. Постройте графовую модель, аналогичную модели в примере 9.4 [см. рис. 9.7 (a)], для произвольной программы и воспользуйтесь ею для оценки среднего времени выполнения и его дисперсии для заданных значений входных данных. Выберите подходящие времена выполнения для различных типов операторов в программе. Можете ли вы с помощью этой модели получить профиль программы?

9.12. Определите чувствительность времени выполнения программы из табл. 9.1 к значениям входных данных, используя модель, построенную в примере 9.5. Вы можете сделать это графически, изобразив функцию $t=t(\mathrm{IENT}, \mathrm{JNT})$, например, как семейство кривых с параметром JNT на плоскости (t, IENT). Вы можете обратиться и к аналитическому методу, определяя чувствительность t к IENT как $\Delta t/\Delta \mathrm{IENT}$ для постоянной JNT. Как чувствительность t к IENT, так и чувствительность к JNT должны, видимо, быть функциями JNT и IENT.

9.13. Удалите оператор 10 (GOTO 10) в программе из табл. 9.1 и модифицируйте Д-карту на рис. 9.9 (a) в соответствии с этим. Выведите уравнения профиля из модифицируемой Д-карты, как описано в примере 9.5, и решите их. Сравните это решение и формулу для времени выполнения, которую из него можно получить, с соответствующими уравнениями и формулой из табл. 9.13.

9.14. Предположим, что программа из табл. 9.1 является модулем (составленным из трех подмодулей) большого пакета программного обеспечения, находящегося в стадии разработки. В конечной версии этой программы операторов 1 и 9 не будет. Входные данные будут передаваться вызванному модулю, а результаты возвращаться вызывающим модулям в качестве параметров. Во время разработки был построен имитатор пакета. В зависимости от типа проводимого исследования для каждого обращения к модулю имитатору необходимо знать некоторые из следующих данных:

a) сколько времени тратится в модуле;
b) сколько выполняется операций ввода-вывода и с какими файлами;
c) сколько раз вызываются различные подмодули (в данном случае MAIN, RK2, FUN).

Как бы вы представили этот модуль в имитаторе?

9.15. Постройте модель локальности на базе ограниченных интервалов локальности, которые представлены на рис. 9.11 (b), для строки из табл. 9.15 (a). Так как существует иерархия интервалов, вам необходимо выбрать такие интервалы, которые, по вашему мнению, наиболее адекватно отображают поведение программы. Конечно, это решение будет зависеть от того, с какой целью строится модель. Например, предположим, что модель должна будет использоваться главным образом для генерации

строк при оценке различных стратегий и что внутри каждого интервала модель стека LRU будет применяться к элементам соответствующего множества активности для получения фактической последовательности ссылок. Выбрав последовательность интервалов, покажите, как вы начнете строить модель и как вы припишите необходимые данные модели стека LRU, которая должна моделировать поведение программы внутри каждого интервала. Специфицируйте в стиле табл. 9.8 алгоритм генератора строки и сформулируйте способ проверки корректности модели. С учетом того, что модель должна представлять поведение строки из табл. 9.15 (а), почему ей отдается предпочтение перед фактической строкой из конкретного описанного выше приложения?

9.16. Оцените экономию времени выполнения программы из табл. 9.1, получаемую в результате модификации, которая описана в примере 9.6, в случае исходных данных из табл 3.9.1. Можете использовать любую аналитическую модель, представленную в разд. 9.3.2 (см. примеры 9.4 и 9.5).

9.17. Программа из табл. 9.1 будет работать на системе, на которой выполнение каждого шага работы оплачивается в соответствии с формулой из задачи 9.2. Вам нужно исследовать эту программу с целью уменьшить стоимость ее выполнения. Опишите, как вы будете действовать, какие используете средства для локализации стоимостных узких мест и какие методы используете для оценки экономии в связи с различными возможными изменениями программы. Не пытаясь вначале получить общие оценки, начните с рассмотрения конкретных случаев модификации программы.

Таблица 3.9.1

Последовательности исходных данных для программы из табл. 9.1

JNT	IENT	JNT	IENT	JNT	IENT
2	7	5	3	1	10
4	5	5	4	2	9
12	8	5	5	3	8
9	10	10	6	4	7
	0	10	7	5	6
		10	8		0
		15	9		
			10		
			0		

9.18. Алгоритм замещения FIFO для системы фиксированного распределения памяти при необходимости заменяет ту страницу из области основной памяти программы, которая находилась в ней дольше остальных с момента последнего выбора ее аппаратом страничной обработки. Определите критический алгоритм переструктурирования FIFO (CFIFO), следуя общему подходу из примера 9.7. Затем примените алгоритм CFIFO к строке ссылок на блоки из табл. 9.4 (а) для $m=2$. Обработайте алгоритмом разбиения на группы матрицу CFIFO при ограничениях, рассмотренных в примере 9.7, и смоделируйте прохождение переструктурированной программы при стратегии FIFO для $m=2$; сделайте то же самое для версий этой программы, переструктурированных в соответствии с алгоритмами близости CLRU и CWS (см. табл. 9.21). Сравните полученные числа страничных отказов.

9.19. Генерирует ли алгоритм переструктурирования CWS меньшее число страничных отказов по сравнению с другими алгоритмами или по сравнению с исходной программой только тогда, когда рабочее множество измеряется во время выполнения в те же моменты, когда оно измерялось по строке алгоритмом CWS? Другими словами, должны ли измерения рабочего множества во время выполнения быть синхронными с измерениями, сделанными алгоритмом CWS, или этот алгоритм работает всегда? Придумайте эксперимент, дающий ответ на этот вопрос, даже если он пригоден для случая только одной программы. Например, можно обрабатывать строку из табл. 9.4 (а), предполагая постоянные интервалы между последовательными ссылками.

Список литературы

Андерсон, Сарджент (Anderson H. A., Jr., Sargent R. G.)
1972 A statistical evaluation of the scheduler of an experimental interactive computing system. In: Freiberger (1972), 73—98.

Арбакл (Arbuckle R. A.)
1966 Computer analysis and thruput evaluation. *Computers and Automation* **15**, 1 (Jan.), 12—15.

Ашенбреннер, Натараджан (Aschenbrenner R. A., Natarajan N. K.)
1971 The Neurotron monitor system. *AFIPS Conf. Proc.* **39** (FJCC), 31—37.

Бадель, Желенб, Лерудье, Потье (Badel M. Gelenbe E., Leroudier J., Potier D.)
1975 Adaptive optimization of a time-sharing system's performance. *Proc. IEEE* **63**, 6 (June), 958—965.

Байю (Bailliu G. R.)
1973 A definition, measure and measurement of the activity of an information processing system. Ph. D. Thesis, University of California, Berkeley (Aug.).

Бард (Bard Y.)
1973 Characterization of program paging in a time-sharing environment. *IBM J. Res. Dev.* **17**, 5 (Sept.), 387—393.

Баскетт, Ченди, Мунц, Паласиос (Baskett F., Chandy K. M., Muntz R. R., Palacios F. G.)
1975 Open, closed and mixed networks of queues with different classes of customers. *J. ACM* **22**, 2 (April), 248—260.

Бейлнер (Beilner H.)
1973 The method of good balance: A tool for improving computing system performance. *Proc. Computer Science and Statistics*: 7*th Annual Symp. on the Interface* (Oct.).

Бейлнер, Вальдбаум (Beilner H., Waldbaum G.)
1972 Statistical methodology for calibrating a trace-driven simulator of a batch computer system. In: Freiberger (1972), 423—459.

Бейзер (Beizer B.)
1970 Analytical techniques for the statistical evaluation of program running time. *AFIPS Conf. Proc.* **37** (FJCC), 519—524.

Бейрд, Джонсон (Baird G. N., Johnson L. A.)
1974 System for efficient program portability. *AFIPS Conf. Proc.* **43** (NCC), 423—429.

Белл (Bell T. E.)
1974 Computer performance variability. *AFIPS Conf. Proc.* **43** (NCC), 761—766.

Белл, Боэм, Уотсон (Bell T. E., Boehm B. W., Watson R. A.)
1972 Framework and initial phases for computer performance improvement. *AFIPS Conf. Proc.* **41** (FJCC), 1141—1154.

Биледи (Belady L. A.)
1966 A study of replacement algorithms for virtual storage computers. *IBM Sys. J.*, **5**, 78—101.

Биледи, Кюнер (Belady L. A., Kuehner C. J.)
1969 Dynamic space sharing in computer systems. *Comm. ACM* **12**, 5 (May), 282.

Бойз, Уорн (Boyse J. W., Warn D. R.)
1975 A straightforward model for computer performance prediction. *Comp. Surveys* **7**, 2 (June), 73—93.

Брайен (Bryan G. E.)
1967 JOSS — 20,000 hours at the console, a statistical summary. *AFIPS Conf. Proc.* **31** (FJCC), 769—777.

Брандвайн, Желенб, Ленфан, Потье (Brandwajn A., Gelenbe E., Lenfant J., Potier D.)
1973 A model of program and system behavior in virtual memory. IRIA-LABORIA, Rocquencourt, France.

Браун Дж., Ченди, Браун Р., Келлер, Тоусли, Диззли (Browne J. C., Chandy K. M., Brown R. M., Keller T. W., Towsley D. F., Dizzly C. W.)
1975 Hierarchical techniques for the development of realistic models of complex computer systems. *Proc. IEEE* **63**, 6 (June), 966—975.

Бринч Хансен (Brinch Hansen P.)
1973 Operating Systems Principles. Prentice-Hall, Englewood Cliffs, N. J.

Бузен (Buzen J. P.)

1971 Analysis of system bottlenecks using a queueing network model. *Proc. ACM-SIGOPS Workshop on System Performance Evaluation*, Harvard University (April), 82—103.

Бузен (Buzen J. P.)
1973 Computational algorithms for closed queueing networks with exponential servers. *Comm. ACM* **16**, 9 (Sept.), 527—531.

Бухгольц (Buchholz W.)
1969 A synthetic job for measuring system performance. *IBM Sys. J.* **8**, 4, 309—318.

Вальдбаум (Waldbaum G.)
1973 Evaluating computing system changes by means of regression models. *Proc. 1st ACM-SIGME Symp. on Measurement and Evaluation* (Feb.), 127—135.

Вилнер (Wilner W. T.)
1972 Design of the Burroughs B1700. *AFIPS Conf. Proc.* **41** (FJCC), 489—497.

Вуд, Форман (Wood D. C., Forman E. H.)
1971 Throughput measurement using a synthetic job stream. *AFIPS Conf. Proc.* **39** (FJCC), 51—56.

Гендерсон, Родригес-Роселл (Henderson G., Rodriguez-Rosell J.)
1974 The optimal choice of window sizes for working set dispatching. *Proc. 2nd ACM-SIGMETRICS Symp. on Measurement and Evaluation* (Oct.), 10—33.

Гибсон (Gibson J. C.)
1970 The Gibson mix. IBM Tech. Rept. TR00. 2043 (June).

Гоел, Лиу (Goel A. L., Liu Y.)
1973 An analysis of ISAM by two-level factorial designs. *Proc. Computer Science and Statistics: 7th Annaual Symp. on the Interface* (Oct.), 16—21.

Голд (Gold M. M.)
1967 Methodology for evaluating time-shared computer usage. Ph. D. Thesis, M. I. T., Cambridge, Mass. Time-sharing and batch-processing: An experimental comparison of their values in a problem-solving situation. *Comm. ACM* **12**, 5 (May 1969), 249—259.

Гордон (Gordon G.)
1969 System Simulation. Prentice-Hall, Englewood Cliffs, N. J.

Гордон (Gordon G.)
1975 The Application of GPSS V to Discrete System Simulation. Prentice-Hall, Englewood Cliffs, N. J.

Гордон, Ньюэлл (Gordon W. J., Newell G. F.)
1967 Closed queueing systems with exponential servers. *Operations Research* **15**, 2 (April), 254—265.

Госден, Сиссон (Gosden J. A., Sisson R. L.)
1962 Standardized comparisons of computer performance. *Information Processing 62* (Proc. IFIP Congress 62). North-Holland, Amsterdam, 57—61.

Грант, Сакман (Grant E. E., Sackman H.)
1966 An exploratory investigation of programmer performance under on-line and off-line conditions. Rept. SP-2581, System Development Corp., Santa Monica, Ca. (Sept.).

Гренандер, Цао (Grenander U., Tsao R. F.)
1972 Quantitative methods for evaluating computer system performance — a review and proposals. In: Freiberger (1972), 3—24.

Гринбаум (Greenbaum H. J.)
1969 A simulator of multiple interactive users to drive a time-shared computer system. Project MAC Tech. Rept. TR-58, M. I. T., Cambridge, Mass. (Jan.).

Гуд, Мун (Good J., Moon B. A. M.)
1972 Evaluating computers for the New Zealand universities. *Datamation* **18**, 11 (Nov.)., 96—99.

Грюнбергер (Gruenberger F.)
1966 Are small, free-standing computers here to stay? *Datamation* **12**, 4 (April), 67—68.

Дал, Нигард (Dahl O., Nygaard K.)
1966 SIMULA: An ALGOL-based simulation language. *Comm. ACM* **9**, 9 (Sept.), 671—678.

Дейкстра (Dijkstra E. W.)
1968 The structure of THE multiprogramming system. *Comm. ACM* **11**, 5 (May), 341—346.

Дейкстра (Dijkstra E. W.)
1972 Notes on structured programming. In: O.-J. Dahl, E. W. Dijkstra, C. A. R. Hoare, Structured Programming. Academic Press, London (1972), 1—82. [Имеется перевод: Дал, Дейкстра, Хоар. Структурное программирование.— М.: Мир, 1975.]

ДеМейс, Вейзер (DeMeis W. M., Weizer N.)
1969 Measurement and analysis of a demand paging time sharing system. Proc. ACM Nat. Conf., 201—216.

Денистон (Deniston W. R.)
1969 SIPE: A TSS/360 software measurement technique. *Proc. ACM Nat. Conf.*, 229—245.

Деннинг (Denning P. J.)
1968a The working set model for program behavior. *Comm. ACM* **11**, 5 (May), 323—333.

Деннинг (Denning P. J.)
1968b Thrashing: Its causes and prevention. *AFIPS Conf. Proc.* **33** (FJCC), 915—922.

Деннинг (Denning P. J.)
1970 Virtual memory. *Comp. Surveys* **2**, 3 (Sept.), 153—189.

Деннинг, Грехем (Denning P. J., Graham G. S.)
1975 Multiprogrammed memory management. *Proc. IEEE*, **63**, 6 (June), 924—939.

Джексон (Jackson J. R.)
1963 Job shop-like queueing systems. *Management Sci.*, **10**, 1, 131—142.

Джослин (Joslin E. O.)
1965 Application benchmarks: The key to meaningful computer evaluations. *Proc. ACM Nat. Conf.*, 27—37.

Джослин (Joslin E. O.)
1968 Computer Selection. Addison-Wesley, Reading, Mass.

Дойч, Грант (Deutsch P., Grant C. A.)
1971 A flexible measurement tool for software systems. *Information Processing 71* (Proc. IFIP Congress 71). North-Holland, Amsterdam.

Драммонд (Drummond M. E., Jr.)
1973 Evaluation and Measurement Techniques for Digital Computer Systems. Prentice-Hall, Englewood Cliffs, N. J. [Имеется перевод: Драммонд М. Е. Методы оценки и измерения дискретных вычислительных систем.— М.: Мир, 1977.]

Дрейпер, Смит (Draper N. R., Smith H.)
1966 Applied Regression Analysis. Wiley, New York.

Желенб (Gelenbe E.)
1973 The distribution of a program in primary and fast buffer storage. *Comm. ACM* **16**, (July), 431—434.

Зальцер, Джинтелл (Saltzer J. H., Gintell J. W.)
1970 The instrumentation of Multics. *Comm. ACM* **13**, 8 (Aug.), 495—500.

Карлсон (Carlson G.)
1971 Hardware monitoring of a software monitor. *Proc. SUM Users Group 1st Annual Meeting* (Jan.).

Каруш (Karush A. D.)
1969 The benchmark method applied to time-sharing systems. Rept. SP-3347, System Development Corp., Santa Monica, Ca. (Aug.).

Каруш (Karush A. D.)
1970 The recording and reduction utility for digital computer systems: Survey and analysis. Rept. SP-3303/000/01. System Development Corp. Santa Monica, Ca. (March)

Керниган, Гамильтон (Kernighan B. W., Hamilton P. A.)
1973 Synthetically generated performance test loads for operating systems. *Proc. 1st ACM-SIGME Symp. on Measurement and Evaluation* (Feb.), 121—126.

Керниган, Плоджер П., Плоджер Д. (Kernighan B. W., Plauger P. J., Plauger D. J.)
1972 On comparing apples and oranges, or, my machine is better than your machine. *Performanve Evaluation Rev.*, **1**, 3 (Sept.) 16—20.

Кивиат (Kiviat P. J.)
1971 Simulation languages. In: Naylor (1971), 406—436.
Кивиат, Виллануэва, Марковиц (Kiviat P. J., Villanueva R., Markowitz H.)
1969 The SIMSCRIPT II Programming Language. Prentice-Hall, Englewood Cliffs, N. J.
Кимблтон (Kimbleton S. R.)
1975 A heuristic approach to computer systems performance improvement. I. A fast performance prediction tool. *AFIPS Conf. Proc.* **44** (NCC), 839—846.
Клейнрок (Kleinrock L.)
1968 Certain analytic results for time-shared processors. *Information Processing 68* (Proc. IFIP Congress 68). North-Holland Amsterdam, 838—845.
Клейнрок (Kleinrock L.)
1975 Queueing Systems, Vol. I: Theory. Wiley, New York. [Имеется перевод: Клейнрок Л. Вычислительные системы с очередями. Т. 1.— М.: Мир, 1975.]
Кнут (Knuth D. E.)
1968 The Art of Computer Programming, Vol. I: Fundamental Algorithms. Addison-Wesley, Reading, Mass. [Имеется перевод: Кнут Д. Искусство программирования для ЭВМ. Т. 1.— М.: Мир, 1976.]
Кнут (Knuth D. E.)
1969 The Art of Computer Programming, Vol. II: Seminumerical Algorithms. Addison-Wesley, Reading, Mass. [Имеется перевод: Кнут Д. Искусство программирования для ЭВМ. Т. 2.— М.: Мир, 1977.]
Кнут (Knuth D. E.)
1971 An empirical study of FORTRAN programs. *Software — Practice and Experience* **1**, 105—133.
Кобаяси (Kobayashi H.)
1974 Application of the diffusion approximation to queueing networks — I: Equilibrium queue distributions. *J. ACM* **21**, 2 (April), 316—328; II: Nonequillibrium distributions and applications to computer modeling. *J. ACM* **21**, 3 (July), 459—469.
Кокрум, Крокетт (Cockrum J. S., Crockett D. E.)
1971 Interpreting the results of a hardware systems monitor. *AFIPS Conf. Proc.* **38** (SJCC), 23—38.
Коленс (Kolence K. W.)
1971 A software view of measurement tools. *Datamation* **17**, 1 (Jan. 1), 32—38.
Коленс (Kolence K. W.)
1972 Software physics and computer performance measurement. *Proc. ACM Nat. Conf.*, 1024—1040.
Коленс, Кивиат (Kolence K. W., Kiviat P. J.)
1973 Software unit profiles and Kiviat figures. *Performance Evaluation Rev.* **2**, 3 (Sept.), 2—12.
Конвей (Conway R. W.)
1963 Some tactical problems in digital simulation. *Management Sci.* **10**, 1 (Oct.), 47—61.
Конвей, Максвелл, Миллер (Conway R. W., Maxwell W. L., Miller L. W.)
1967 Theory of Scheduling. Addison-Wesley, Reading, Mass.
Коффман (Coffman E. G., Jr.)
1969 Analysis of a drum input/output queue under scheduled operation in a paged computer systems. *J. ACM* **16**, 1 (Jan.), 73—90.
Коффман, Деннинг (Coffman E. G., Jr., Denning P. J.)
1973 Operating Systems Theory. Prentice-Hall, Englewood Cliffs, N. J.
Куперман, Линч, Тецлафф (Cooperman J. A., Lynch H. W., Tetzlaff W. H.)
1972 SGP — an effective use of performance and usage data. *Computer* **5**, 5 (Sept.—Oct.), 20—23.
Куртуа (Courtois P. J.)
1971 On the near-decomposability of networks of queues and of stochastic models of multiprogramming computing systems. Rep. CMU-CS-72-111, Computer Science Dept., Carnegie-Mellon University, Pittsburgh (Nov.).

Куртуа (Courtois P. J.)
1975 Decomposability, instabilities, and saturation in multiprogramming systems. *Comm. ACM* **18**, 7 (July), 371—376.
Кэмпбелл, Хеффнер (Campbell D. J., Heffner W. J.)
1968 Measurement and analysis of large operaring systems during system development. *AFIPS Conf. Proc.* **33** (FJCC), 903—914.
Лассетер, Ло, Ченди, Брауни (Lasseter G. L., Lo T., Chandy K. M., Browne J. C.)
1973 Statistical and pattern based models for CPU burst prediction. *Proc. Computer Science and Statistics: 7th Annual Symp. on the Interface* (Oct.), 123—129.
Линдсей (Lindsay D. S.)
1975 A study in operating system performance measurement and modeling. Ph. D. Thesis, University of California, Berkeley (June).
Линч (Lynch W. C.)
1972 Operating system performance. *Comm. ACM* **15**, 7 (July), 579—585.
Литтл (Little J. D. C.)
1961 A proof for the queueing formula L-λ W. *Operations Research* 9, 3 (May), 383.
Лукас (Lucas H. C., Jr.)
1971a Synthetic program specifications for performance evaluation. Research Paper No. 33, Graduate School of Business, Stanford University, Stanford, Ca. (Aug.).
Лукас (Lucas H. C., Jr.)
1971b Performance evaluation and monitoring. *Comp. Surveys* **3**, 3 (Sept.), 79—91.
Льюис, Ю (Lewis P. A. W., Yue P. C.)
1972 Statistical analysis of series of events in computer systems. In: Freiberger (1972), 265—280.
Мак-Дугалл (MacDougall M. H.)
1970 Computer system simulation: An introduction. *Comp. Surveys* **2**, 3 (Sept.), 191—209.
Мак-Кинни (McKinney J. M.)
1969 A survey of analytical time-sharing models. *Comp. Surveys* **1**, 2 (June), 105—116.
Мартин (Martin J.)
1967 Design of Real-Time Computer Systems. Prentice-Hall, Englewood Cliffs, N. J.
Маттсон, Джекси, Слутц, Трейгер (Mattson R. L., Gecsei J., Slutz D. R., Traiger I.L.)
1970 Evaluation techniques for storage hierarchies. *IBM Sys. J.*, 9, 2, 78—117.
Морган, Кэмпбелл (Morgan D. E., Campbell J. A.)
1973 An answer to a user's plea? *Proc. 1st ACM-SIGME Symp. on Measurement and Evaluation* (Feb.), 112—120.
Мунц (Muntz R. R.)
1975 Analytic modeling of interactive systems. *Proc. IEEE* **63**, 6 (June), 946—953.
Мэдисон, Батсон (Madison A. W., Batson A. P.)
1976 Characteristics of program localities. *Comm. ACM* **19**, 5 (May), 285—294.
Найт (Knight K. E.)
1963 A study of technological innovation: The evolution of digital computers. Ph. D. Thesis, Carnegie Institute of Technology, Pittsburgh, (Nov.).
Найт (Knight K. E.)
1968 Evolving computer performance, 1962—1967. *Datamation* **14**, 1 (Jan.), 31—35.
Нейлор (Naylor T.)
1971 Computer Simulation Experiments with Models of Economic Systems. Wiley, New York. [Имеется перевод: Машинные имитационные эксперименты с моделями экономических систем.— М. «Мир», 1975.]
Нильсен (Nielsen N. R.)
1967 The simulation of time-sharing systems. *Comm. ACM* **10**, 7 (July), 397—412.
Нильсен (Nielsen N. R.)
1971 An analysis of some time-sharing techniques. *Comm. ACM* **14**, 2 (Feb.), 79—90.
Ноу, Натт (Noe J. D., Nutt G. J.)
1972 Validation of a trace-driven CDC 6400 simulation. *AFIPS Conf. Proc.* **40** (SJCC), 749—757.
Ньюэлл (Newell G. F.)
1971 Applications of Queueing Theory. Chapman and Hall, London, Ch. 6.

Оливер, Бейрд, Кук, Джонсон, Хойт (Oliver P., Baird G., Cook M., Johnson A., Hoyt P.)
1974 An experiment in the use of synthetic programs for system benchmarks. *AFIPS Conf. Proc.* **43** (NCC), 431—438.
Парнас (Parnas D. L.)
1969 More on simulation languages and design methodology for computer systems. *AFIPS Conf. Proc.* **34**, (SJCC), 739—743.
Парнас (Parnas D. L.)
1972 On the criteria to be used in decomposing systems into modules. *Comm. ACM* **15**, 12 (Dec.), 1053—1062.
Парнас, Сивиорек (Parnas D. L., Siewiorek D. P.)
1975 Use of the concept of transparency in the design of hierarchically structured systems. *Comm. ACM* **18**, 7 (July), 401—408.
Парупуди, Виноград (Parapudi M., Winograd J.)
1972 Interactive task behavior in a time-shared environment. *Proc. ACM Nat. Conf.*, **2** (Aug.), 680—692.
Питерсон (Peterson T. G.)
1974 A comparison of software and hardware monitors. *Performance Evaluation* Rev. **3**, 2 (June), 2—5.
Помрой (Pomeroy J. W.)
1972 A quide to programming tools and techniques. *IBM Sys. J.*, **11**, 3, 234—254.
Прив (Prieve B. G.)
1973 A page partition replacement algorithm. Ph. D. Thesis, University of California, Berkeley (Dec.).
Притскер (Pritsker A. A.)
1974 The GASP IV Simulation Language. Wiley, New York.
Рамамурти (Ramamoorthy C. V.)
1965 Discrete Markov analysis of computer programs. *Proc. ACM Nat. Conf.*, 386—392.
Ранделл (Randell B.)
1968 Computer and operating system modeling. Draft of a lecture given at the University of Utah, Salt Lake City, Jan. 25.
Раселл, Эстрин (Russell E. C., Estrin G.)
1969 Measurement based automatic analysis of FORTRAN programs. *AFIPS Conf. Proc.* **34** (SJCC), 723—732.
Рейчелсон, Коллинз (Raichelson E., Collins G.)
1964 A method for comparing the internal operating speeds of computers. *Comm. ACM* **7**, 5 (May), 309—310.
Рехтшаффен (Rechtschaffen R. N.)
1972 Queueing simulation using a random number generator. *IBM Sys. J.*, **11**, 3, 255—271.
Роек, Эмерсон (Roek D. J., Emerson W. C.)
1969 A hardware instrumentation approach to evaluation of a large scale system. *Proc. ACM Nat. Conf.*, 351—367.
Розвадовски (Rozwadowski R. T.)
1973 A measure for the quantity of computation. Proc. *1st ACM — SIGME Symp. on Measurement and Evaluation* (Feb.), 100—111.
Саал, Шустек (Saal H. J., Shustek L. J.)
1972 Microprogrammed implementation of computer measurement techniques. *Proc. 5th annual Workshop on Microprogramming* (Sept.).
Сакман (Sackman H.)
1964 Regenerative recording in man-machine digital systems, *Proc. Nat. Winter Convention on Military Electronics* (Feb.), 16. 14—16.19.
Сакман (Sackman H.)
1968 Time-sharing versus batch processing: The experimental evidence. *AFIPS Conf. Proc.* **32** (SJCC), 1—10.

Саммет (Sammet J. E.)
1969 Programming Languages: History and Fundamentals. Prentice-Hall, Englewood Cliffs, N. J.
Свободова (Svobodova L.)
1973 Online system performance measurements with software and hybrid monitors. *Operating Systems Rev.* **7**, 4 (Oct.), 45—53.
Седжвик, Стоун, Мак-Дональд (Sedgewick R., Stone R., McDonald J. W.)
1970 SPY — a program to monitor OS/360. *AFIPS Conf. Proc.* **37** (FJCC), 119—128.
Сейр (Sayre D.)
1969 Is automatic folding of programs efficient enough to displace manual? *Comm. ACM* **12**, 12 (Dec.), 656—660.
Секино (Sekino A.)
1973 Throughput analysis of multiprogramming virtual memory computer systems. *Proc. 1st ACM — SIGME Symp. on Measurement and Evaluation* (Feb.), 47—53.
Симен, Соуси (Seaman P. H., Soucy R. C.)
1969 Simulating operating systems. *IBM Sys. J.*, **8**, 4, 264—279.
Симс (Syms G. H.)
1974 Benchmarked comparison of terminal support systems for IBM 360 computers. *Performance Evaluation Rev.* **3**, 2 (June), 6—34.
Смит (Smith L. B.)
1967 A comparison of batch processing and instant turnaround. *Comm. ACM* **10**, 8 (Aug.), 495—500.
Снедекор, Кокрен (Snedecor G. W., Cochran W. G.)
1967 Statistical Methods, 6th ed., Iowa State University Press, Ames.
Спирн, Деннинг (Spirn J. R., Denning P. J.)
1972 Experiments with program locality. *AFIPS Conf. Proc.* **41**, (FJCC), 611—621.
Сринивасан, Клейнман (Sreenivasan K., Kleinman A. J.)
1974 On the construction of a representative synthetic workload. *Comm. ACM* **17**, 3 (Mar.), 127—133.
Страус (Strauss J. C.).
1972 A benchmark study. *AFIPS Conf. Proc.* **41** (FJCC), 1225—1233.
Стритер (Streeter D. N.)
1973 Centralization or dispersion of computing facilities. *IBM Sys. J.*, **12**, 3, 283—301.
Тейчроев, Любин (Teichroew D., Lubin J. F.)
1966 Computer simulation: Discussion of the technique and comparison of languages. *Comm. ACM* **9**, 10 (Oct.), 723—741.
Тиммрек (Timmreck E. M.)
1973 Computer selection methodology. *Comp. Surveys* **5**, 4 (Dec.), 199—222.
Трейгер, Маттсон (Traiger I. L., Mattson R. L.)
1972 The evaluation and selection of technologies for computer storage systems. IBM Research Rept. RJ 967 (Feb.)
Уилкс (Wilkes M. V.)
1971 Automatic load adjustment in time-sharing systems. *Proc. ACM — SIGOPS Workshop on System Performance Evaluation*, Harvard University (April), 308—320.
Уотсон (Watson R. W.)
1970 Timesharing System Design Concepts. McGraw-Hill, New York.
Феллер (Feller W.)
1968 An Introduction to Probability Theory and Its Applications, Vol. I, 3rd ed. Wiley, New York. [Имеется перевод 2 изд.: Феллер В. Введение в теорию вероятностей и ее приложения.— М.: Мир, 1964.]
Феничел, Гроссман (Fenichel R. R., Grossman A. J.)
1969 An analytic model of multiprogrammed computing. *AFIPS Conf. Proc.* **34** (SJCC), 717—721.
Феррари (Ferrari D.)
1972 Workload characterization and selection in computer performance measurement. *Computer* **5**, 4 (July-Aug.), 18—24.

Феррари (Ferrari D.)
1973 Architecture and instrumentation in a modular interactive system. *Computer* **6**, 11 (Nov.), 25—29.
Феррари (Ferrari D.)
1974a Improving program locality by strategy-oriented restructuring. *Information Processing 74* (Proc. IFIP Congress 74). North-Holland, Amsterdam, 266—270.
Феррари (Ferrari D.)
1974b Improving locality by critical working sets. *Comm. ACM* **17**, 11 (Nov), 614—620.
Феррари (Ferari D.)
1975 Tailoring programs to models of program behavior. *IBM J. Res. Dev.* **19**, 3 (May), 244—251.
Феррари, Лиу (Ferrari D., Liu M.)
1975 A general-purpose software measurement tool. *Software — Practice and Experience*, **5**, 2, 181—192.
Феррари, Линдсей, Пик (Ferrari D., Lindsay D. S., Peak P. M.)
1975 Variance reduction techniques in computer system performance measurement. *Proc. Int. Comp. Symp. 75*, North-Holland, Amsterdam, 185—189.
Фишман (Fishman G. S.)
1973 Concepts and Methods in Discrete Event Digital Simulation. Wiley, New York.
Флинн (Flynn M. J.)
1974 Trends and problems in computer organizations. *Information Processing 74* (Proc. IFIP Congress 74). North-Holland, Amsterdam, 3—10.
Фогель, Виноград (Fogel M., Winograd J.)
1972 EINSTEIN: An internal driver in a time-sharing environment. *Operating Systems* Rev. **6**, 3 (Oct.), 6—14.
Фоули (Foley J. D.)
1971 An approach to the optimum design of computer graphics systems. *Comm. ACM* **14**, 6 (June), 380—390.
Фрейбергер (Freiberger W., ed.)
1972 Statistical Computer Performance Evaluation. Academic Press, New York.
Фукс, Джексон (Fuchs E., Jackson P. E.)
1970 Estimates of distributions of random variables for ceratin computer communications traffic models. *Comm. ACM* **13**, 12 (Dec.), 752—757.
Фуллер (Fuller S. H.)
1972 An optimal drum scheduling algorithms. *IEEE Trans. on Computers* **C-21**, 11 (Nov.), 1153—1165.
Фуллер, Сван, Вулф (Fuller S. H., Swan R. J., Wulf W. A.)
1973 The instrumentation of C. mmmp, a multi-(mini)processor. *Proc. COMPCON 73* (Feb.), 173—176.
Хант, Диер, Гарнац (Hunt E., Diehr G., Garnatz D.)
1971 Who are the users? An analysis of computer use in a university computer center. *AFIPS Conf. Proc.* **38** (SJCC), 231—238.
Хеллерман, Смит (Hellerman H., Smith H. J.)
1970 Throughput analysis of some idealized input, output and compute overlap configurations. *Comp. Surveys* **3**, 2 (June), 111—118.
Хеллерман (Hellerman L.)
1972 A measure of computational work. *IEEE Trans. on Computers* **C-21** 5 (May).
Хиллигасс (Hillegass J. R.)
1966 Standardized benchmark problems measure computer performance. *Computers and Automation* **15**, 1 (Jan.), 16—19.
Холтвик (Holtwick G. M.)
1971 Designing a commercial performance measurement system. *Proc. ACM-SIGOPS Workshop on System Performance Evaluation*, Harvard University (April), 29—58.
Хьюз, Кроншоу (Hughes J., Cronshaw D.)
1973 On using a hardware monitor as an intelligent peripheral. *Performance Evaluation Rev.* **2**, 4 (Dec.), 3—19.
Хьюз, Мо (Hughes P. H., Moe G.)

1973 A structural approach to computer performance analysis. *AFIPS Conf. Proc.* **42** (NCC), 109—120.
Хэтфилд, Джеральд (Hatfield D. J., Gerald J.)
1971 Program restructuring for virtual memory. *IBM Sys. J.* **10**, 3, 168—192.
Цао, Марголин (Tsao R. F., Margolin B. H.)
1972 A multi-factor paging experiment — II. Statistical methodology. In: Freiberger (1972), 135—158.
Цао, Комо, Марголин (Tsao R. F., Comeau L. W., Margolin B. H.)
1972 A multi-factor paging experiment — I. The experiment and the conclusions. In: Freiberger (1972), 103—134.
Цюрхер, Ранделл (Zurcher F. W., Randell B.)
1968 Iterative multi-level modelling — a methodology for computer system design. *Information Processing 68* (Proc. IFIP Congress 68). North-Holland, Amsterdam.
Чемберлен, Фуллер, Лиу (Chamberlin D. D., Fuller S. H., Liu L. Y.)
1973 An analysis of page allocation strategies for multiprogramming systems with virtual memory. *IBM J. Res. Dev.* **17**, 5 (Sept.), 404—412.
Ченг (Cheng P. S.)
1969 Trace-driven system modeling. *IBM Sys. J.*, **8**, 4, 280—289.
Червены, Найт (Cerveny R. P., Knight K. E.)
1973 Performance of minicomputers. *Proc. 2nd Texas Conf. on Computer Systems* (Nov.), 28.1—28.7.
Чу, Опдербек (Chu W. W., Opderbeck H.)
1972 The page fault frequency replacement algorithm. *AFIPS Conf. Proc.* **41** (FJCC), 597—609.
Шарп (Sharpe W. F.)
1969 The Economics of Computers. Columbia University Press, New York.
Шатцофф, Цао, Виг (Schatzoff M., Tsao R. F., Wiig R.)
1967 An experimental comparison of time sharing and batch processing. *Comm. ACM* **10**, 5 (May), 261—265.
Шветман, Браун (Schwetman H. D., Browne J. C.)
1972 An experimental study of computer system performance. *Proc. ACM Nat. Conf.*, 693—703.
Шерман, Баскетт, Браун (Sherman S., Baskett III, F., Browne J. C.)
1972 Trace-driven modeling and analysis of CPU scheduling in a multiprogramming system. *Comm. ACM* **15**, 12 (Dec.), 1063—1069.
Шерр (Scherr A. L.)
1967 An Analysis of Time-Shared Computer Systems. M. I. T. Press, Cambridge, Mass.
Шнейдевинд (Schneidewind N. F.)
1966 Analytic model for the design and selection of electronic digital computers. Ph. D. Thesis, University of Southern California, Los Angeles (Jan.).
Шоуп, Кашмарак, Ингрем, Декер (Shope W. L., Kashmarak K. L., Inghram J. W., Decker W. F.)
1970 System performance study. *Proc. SHARE* **34**, 1 (Mar.), 439—530.
Штимлер (Stimler S.)
1969 Real-Time Data-Processing Systems. McGraw-Hill, New York.
Шульман (Schulman F. D.)
1967 Hardware measurements device for IBM System/360 time sharing evaluation. *Proc. ACM Nat. Conf.*, 103—109.
Эппл (Apple C. T.)
1965 The program monitor — a device for program performance measurement. *Proc. ACM Nat. Conf.*, 66—75.
Эриксон (Erikson W. J.)
1966 A pilot study of interactive versus noninteractive debugging. Rept. TM-3296, System Development Corp., Santa Monica, Ca. (Dec.).
Эстрин, Гопкинс, Когган, Крокер (Estrin G., Hopkins D., Coggan B., Crocker S. D.)
1967 SNUPER COMPUTER — a computer in instrumentation automaton. *AFIPS Conf. Proc.* **30** (SJCC), 645—656.

ОГЛАВЛЕНИЕ

От редактора и переводчиков 5
Предисловие 8

1. Подход к оценке производительности 13

1.1. Введение 13
1.2. Объекты работ по оценке 15
1.3. Этапы работы по оценке 22
1.4. Индексы производительности 24
1.5. Классификация работ по оценке 31
1.6. Обзор методов оценки 32
Задачи 36

2. Методы измерения 39

2.1. Работы по измерению 39
2.2. Введение в измерители 42
2.3. Аппаратные измерители 45
2.3.1. Введение 45
2.3.2. Фиксированные аппаратные измерители 46
2.3.3. Измерители с фиксированной программой 48
2.3.4. Измерители с запоминаемой программой 54
2.4. Программные и микропрограммные измерители 58
2.4.1. Введение 58
2.4.2. Внутренне управляемые измерители 61
2.4.3. Внешне управляемые (выборочные) измерители 71
2.5. Измерительные системы 80
2.6. Планирование измерительных экспериментов 82
2.6.1. Постановка задачи 82
2.6.2. Идентификация факторов 84
2.6.3. Выбор уровней для каждого сеанса 86
2.6.4. Время и длительность сеанса 90
2.7. Интерпретация результатов измерений 97
2.7.1. Сжатие данных и методы описания 97
2.7.2. Дисперсионный анализ 100
2.7.3. Регрессионный анализ 107
Задачи 113

3. Методы моделирования 117

3.1. Работы по моделированию 117
3.2. Формулировка и конструирование модели 124
3.2.1. Основные проблемы 124
3.2.2. Структура модели 127
3.2.3. Другие организации моделей 137
3.2.4. Моделирование рабочей нагрузки 141
3.2.5. Реализация моделирующей программы 149
3.3. Калибровка и проверка пригодности модели 152
3.3.1. Постановка задачи 152

3.3.2. Калибровка моделирующей программы 157
3.3.3. Методика калибровки 159
3.4. Планирование имитационных экспериментов 164
3.4.1. Планирование эксперимента при моделировании 164
3.4.2. Продолжительность прогона модели 166
3.5. Интерпретация результатов моделирования 174
Задачи 175

4. Аналитические методы 178

4.1. Аналитические исследования 178
4.2. Детерминированные модели 182
4.2.1. Введение 182
4.2.2. Модель перекрытия работы центрального процессора и ввода-вывода 183
4.2.3. Модель средних значений для мультипрограммной системы . . 188
4.3. Вероятностные модели 191
4.3.1. Марковские модели и модели с очередями 191
4.3.2. Непрерываемые модели с единственным центром обслуживания . 200
4.3.3. Прерываемые модели с одним центром обслуживания 211
4.3.4. Модели — открытые сети 217
4.3.5. Модели — замкнутые сети 223
4.4. Калибровка и контроль аналитической модели 235
Задачи 238

5. Описание рабочей нагрузки 242

5.1. Введение 242
5.2. Формулировка модели рабочей нагрузки 246
5.2.1. Подходы к описанию рабочей нагрузки 246
5.2.2. Проблема представительности 249
5.2.3. Проблема системно-независимого описания 253
5.3. Построение моделей рабочих нагрузок 260
5.3.1. Основные определения 260
5.3.2. Естественные модели рабочих нагрузок 260
5.3.3. Неисполняемые искусственные модели 264
5.3.4. Исполняемые искусственные модели 268
5.3.5. Построение исполняемой искусственной модели 279
5.4. Калибровка и контроль модели рабочей нагрузки 296
Задачи 299

6. Оценка производительности в задачах выбора 302

6.1. Задачи выбора 302
6.2. Методы выбора 305
6.2.1. Процедура конкурсного выбора 305
6.2.2. Роль оценки производительности в процедурах выбора 308
6.3. Сравнение производительностей 312
6.3.1. Задача сравнения 312
6.3.2. Сравнение режимов обработки 316
6.3.3. Сравнение поставщиков 321
6.3.4. Сравнение установок 335
6.3.5. Сравнение компонентов системы 341
6.4. Методы оценки для работ по выбору 344
6.5. Описание рабочей нагрузки для работ по выбору 345
Задачи 356

7. Оценка производительности в задачах усовершенствования 360

7.1. Задачи усовершенствования 360
7.2. Методы усовершенствования 370
7.3. Диагностика 370
7.3.1. Основная задача 370
7.3.2. Аналитическое исследование узких мест 378
7.3.3. Обнаружение узких мест 386
7.3.4. Статистический метод диагностики 396
7.4. Терапия 398
7.4.1. Настроечная терапия 398
7.4.2. Процедуры самонастройки 409
7.4.3. Методы развития 316
7.5. Методы оценки в работах по усовершенствованию 421
7.6. Описание рабочей нагрузки в работах по усовершенствованию 422
Задачи 424

8. Оценка производительности в задачах проектирования 427

8.1. Задачи проектирования 427
8.2. Методы проектирования 431
8.3. Проблемы производительности в итеративном проектировании . . . 435
8.3.1. Аналитические методы проектирования первоначальной конфигурации 435
8.3.2. Проектирование иерархии памяти 443
8.3.3. Исследование компромиссов проектирования 456
8.4. Факторы производительности в иерархическом проектировании . . . 464
8.5. Методы оценки в проектировании 474
8.6. Описание рабочей нагрузки для проектных работ 477
Задачи 480

9. Оценка производительности программ 484

9.1. Введение 484
9.2. Индексы производительности программы 486
9.2.1. Время и память 486
9.2.2. Индексы поведения программы 496
9.3. Методы оценки программ 506
9.3.1. Измерение программ 506
9.3.2. Моделирование программ 519
9.4. Проблемы оценки программ 544
9.4.1. Выбор программ 544
9.4.2. Усовершенствование программ 546
9.4.3. Разработка программ 558
Задачи 561

Список литературы 565